




# Norme API Groupe CA

## Spécifications détaillées

Référentiel d'architecture d'entreprise Groupe CA

<i>Catégorie</i>	Normes d'interopérabilité
<i>Type</i>	Spécifications détaillées
<i>Libellé</i>	Exposition & usage d'API
<i>Version</i>	1.2
<i>Date d'émission</i>	04/2019
<i>Concepteur</i>	CESI – Cellule d'expertise des standards d'interopérabilité Groupe
<i>Rédacteur</i>	CASA/ITD/AEG – Marc Helloco

<i>Identifiant</i>	AE01
<i>Nom</i>	RefAE_NormeAPI
<i>Version étendue (avec révisions mineures)</i>	1.2.016 (14/11/2023)
<i>Modifications par versions</i>	<a href="#">ReleaseNotes.txt</a>
<i>Autre Version</i>	<a href="#">English Version</a> 

Version	Date	Paragraphes Modifiés/Ajoutés	Objet
1.0	14/12/2017	N/A	Création du document
1.1	15/02/2018	tous	Adaptation pour prise en compte de cas d'usage Groupe supplémentaires
1.2	09/04/2019	tous	Adaptation/simplification suite aux retours des premiers projets

### Suivi des validations

Version	Date	Organe de revue / validation	Observations
1.0	19/12/2017	CAEG	-
1.1	06/03/2018	CAEG	
1.2	09/04/2019	CAEG	

## Table des matières

1	Introduction .....	6
1.1	Contexte .....	6
1.2	Finalité du document .....	6
1.3	Périmètre d'application de la norme .....	6
1.4	Personnes concernées .....	7
1.5	Contacts .....	7
2	Principes généraux .....	8
2.1	Définitions .....	8
2.1.1	Ressources .....	8
2.1.2	API .....	8
2.1.3	Rôles .....	8
2.2	Principaux cas d'usage .....	9
2.2.1	Cas d'usage impliquant 1 entité .....	9
2.2.2	Cas d'usage impliquant 2 entités .....	9
2.2.3	Cas d'usage impliquant 3 entités .....	10
2.3	Architecture des échanges (stateless) .....	11
2.4	Niveau de maturité REST .....	11
3	Définition des API .....	14
3.1	Conception .....	14
3.1.1	Spécification fonctionnelle .....	14
3.1.2	Fichier Swagger .....	14
3.1.3	Format des uri .....	15
3.1.4	Données d'en-tête .....	17
3.2	Publication et exposition .....	20
3.2.1	Gestion des API (API Management) .....	20
3.2.2	Gestion des consommateurs .....	22
3.2.3	Mise à disposition d'une Sandbox .....	23
3.2.4	Portail API fédérateur Groupe .....	24
3.3	Cycle de vie .....	24
3.3.1	Généralités .....	24
3.3.2	Montée de version .....	25
3.3.3	Nomenclature .....	27
3.3.4	Gouvernance .....	27

4	Définition des ressources .....	30
4.1	Généralités .....	30
4.2	Conception des ressources .....	30
5	Sécurisation des échanges.....	31
5.1	Exigences générales .....	31
5.1.1	Authentification de l'application consommatrice .....	32
5.1.2	Authentification de l'utilisateur (selon cas d'usage) .....	32
5.1.3	Habilitations fonctionnelles de l'utilisateur .....	32
5.1.4	Contrôle de la souscription à l'API.....	33
5.1.5	Contrôle de l'accès à la ressource .....	33
5.1.6	Autres données applicatives : application des exigences générales de sécurité (SEC-API) 34	
5.2	Mise en œuvre des mesures de sécurité .....	35
5.2.1	Bonnes pratiques de développement .....	35
5.2.2	Protection des données échangées.....	38
5.2.3	Chiffrement des données sensibles.....	40
5.2.4	Contrôles à réaliser sur les éléments échangés .....	42
5.2.5	Gestion des certificats .....	44
5.3	Authentification et autorisation de l'application consommatrice et de l'utilisateur pour l'accès à une API 45	
5.3.1	OAuth2 et OpenID Connect (OIDC) .....	45
5.3.2	Rôles .....	46
5.3.3	Types de jetons intervenants dans OAuth et OpenID Connect.....	47
5.3.4	Méthodes d'authentification des applications consommatrices clientes.....	48
5.3.5	Choix des cinématiques (ou « Grant Types ») .....	51
5.3.6	Authentification graduée .....	108
5.4	SSO utilisateur et lancement d'API .....	109
5.4.1	Cas des applications accédées via jeton JWT (NNI IHM ou autres).....	109
5.4.2	Cas d'usage d'appel d'une API depuis l'application cible.....	112
5.4.3	Cas d'usage d'appel d'une API depuis une autre API de l'entité cible .....	112
5.4.4	Cas d'usage de la transmission d'identité entre plusieurs API internes à une entité ...	112
5.4.5	Autres cas possible de SSO entre applications via OpenID Connect.....	112
6	Cas dérogatoires à l'utilisation de la spécification.....	113
6.1	Mise en œuvre de l'API DSP2.....	113

6.2	Mise en œuvre d'une API Webhook .....	114
7	Annexes .....	115
7.1	Documents de référence .....	115

## 1 Introduction

### 1.1 Contexte

La montée en puissance des terminaux mobiles et l'évolution de l'architecture des applications Web ont mis en évidence la nécessité de compléter la palette des normes et standards de l'architecture d'échanges issus des travaux de l'AEG.

On note ainsi au travers des applications plébiscitées par les utilisateurs un déport de la logique de présentation, traditionnellement réalisée sur un serveur, vers le poste de travail pour une meilleure interaction avec l'utilisateur (application client riche). Les appels de services, auparavant réalisés entre serveur de présentation et serveur métier, peuvent maintenant être émis directement depuis le terminal.

Dans ce contexte, il est indispensable de s'appuyer sur un socle technique permettant l'appel de services aussi bien depuis un terminal que depuis un serveur, tout en garantissant la sécurité des échanges.

Les enjeux associés sont :

- Développer des applications plus réactives, des IHM plus riches,
- Mettre en œuvre des architectures techniques plus simples et plus efficaces,
- Ouvrir plus facilement les SI à des entités du Groupe Crédit Agricole,
- Ouvrir plus facilement les SI à des partenaires extérieurs au Groupe Crédit Agricole.

### 1.2 Finalité du document

Le présent document restitue l'ensemble des principes standardisés d'exposition et d'usage des API sur un terminal ou sur un serveur.

**Il constitue une spécification référentielle d'exposition de ressources au moyen d'API à des entités du groupe Crédit Agricole ainsi qu'à des partenaires extérieurs au Groupe.**

**Concernant les échanges internes à une même entité il constitue une préconisation qui ne revêt pas de caractère obligatoire.**

### 1.3 Périmètre d'application de la norme

Les travaux couvrent :

- Les modalités d'échanges entre entités du groupe Crédit Agricole au moyen d'API REST
- Les modalités d'échange pour des API REST du Groupe fournies à des partenaires extérieurs au Groupe

**Pour les API Internet ou Intranet accédées uniquement par des applicatifs de la même entité (intra-entité), la norme API n'a pas de caractère obligatoire mais constitue simplement une bonne pratique.**

Cela comprend :

- La définition des API par les entités du Groupe,

- La publication des API par les entités du Groupe,
- L'enrôlement des applications consommatrices aux API exposées par les entités du Groupe,
- Les principes d'identification, authentification, habilitations de l'acteur.

Les modalités de consommation par des applications du Groupe d'API exposées par des sociétés extérieures au Groupe est hors périmètre.

Cette spécification s'applique également aux échanges de serveur à serveur dépréciant les spécifications NNIWS et NNIREST.

La définition des exigences de sécurité associées à ces échanges et des mesures à mettre en œuvre pour les satisfaire sont issues de travaux du groupe de travail SECAPI (Sécurité Applicative) via la réalisation d'une analyse de risque MESARI sur les API REST [cf. [ARAPI](#)]. Le GT SECAPI est coordonné par le bureau d'étude de la sécurité Groupe (CASA/ITD/SRI) et officie sous l'égide du Comité Politiques et Standards SSI (anciennement nommé CPST).

**Les choix de mise en œuvre des composants sont hors périmètre.** Chaque entité est libre d'utiliser une solution qui lui est propre pour autant qu'elle respecte les normes et standards en vigueur dans le groupe.

### 1.4 Personnes concernées

Ce document s'adresse aux architectes ou chefs de projets qui ont à mettre en œuvre l'exposition ou l'invocation d'une API et de ses ressources.

### 1.5 Contacts

Ce document s'inscrit dans le "Cadre Référentiel de l'Architecture d'Entreprise Groupe", et plus particulièrement dans le Référentiel des échanges pour ouvrir plus facilement les SI à des entités du Groupe Crédit Agricole et aux partenaires extérieurs. Il peut être consulté sur le site Intranet de l'Architecture d'Entreprise Groupe à l'adresse indiquée en [\[REC\]](#).

Pour obtenir des précisions sur les points abordés dans ce document et leur mise en œuvre ou plus largement sur les aspects architecture applicative et technique vous pouvez contacter l'Architecture Entreprise Groupe (Crédit Agricole SA / ITD / AEG).

## 2 Principes généraux

### 2.1 Définitions

#### 2.1.1 Ressources

On considère comme ressource un objet dont l'exposition à un consommateur est justifiée par un besoin métier d'accès unitaire et dont l'état évolue en fonction des actions qu'il subit :

- Pour chaque action, il existe un état initial de l'objet (qui sera donc une condition préalable pour l'exécution de l'action), et un état final (qui sera donc le résultat de l'action).
- Toute action sur l'objet est décrite par un verbe (GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS), utilisé pour invoquer la ressource dans son état initial et la faire passer à l'état final (i.e. : obtenir son résultat)

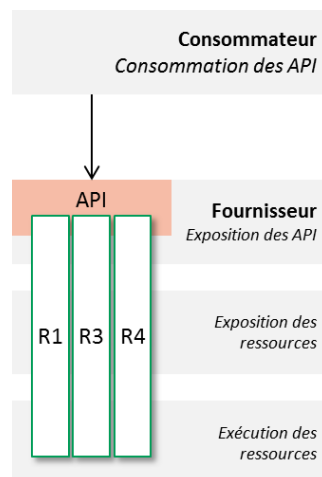
L'accès à ces ressources s'effectue à travers des liens URI. Un URI doit permettre d'identifier une ressource de manière permanente et univoque.

Le format de ces URI est décrit au [chapitre 3.1.3](#).

#### 2.1.2 API

Les APIs sont définies dans le cadre de référence d'une architecture REST de la manière suivante :

- Une API (Application Programming Interface) est une enveloppe de ressource(s) (une ou plusieurs),
- Une API représente une interface permettant de piloter l'accès à ces ressources et permet de fournir un certain niveau d'abstraction au consommateur en masquant la complexité d'accès au SI,
- Les APIs sont référencées et publiées dans un catalogue pour ensuite être consommées par des applications internes et/ou externes,
- Les applications souscrivent à une API pour avoir les habilitations pour la consommer.



Pour compléter cette définition du terme API, l'Architecture d'Entreprise Groupe a publié un [Glossaire API](#) dans lequel est notamment défini la notion d'**API Digital qui constitue la cible du Groupe en terme de conception d'API**. Une API Digital y est décrite comme étant générique, documentée, testable, avec une souscription en self-service, disposant d'un portail développeur et d'un portail consommateur, respectant la présente Norme API, etc.



Les entités peuvent être amenées à mettre à disposition une même ressource au moyen de différentes API avec des modalités d'exposition différentes.

A noter qu'il existe un autre type d'API présente dans la norme d'échange par message asynchrone [NASYNC], il s'agit des **API Webhook**. Elles ne répondent pas à la même définition que l'API digitale. Les API Webhook sont décrites au §6.2.

## 2.1.3 Rôles

La réalisation d'échange inter-applicatif au moyen d'API implique 3 types d'objets :

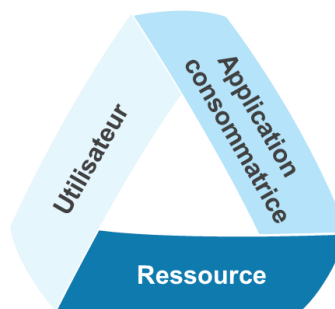
### Application consommatrice

Représente l'application à l'origine de l'échange, hébergée par l'entité qui en a la responsabilité. Il peut s'agir :

- D'une application native (desktop ou mobile),
- D'une application web exécutée sur un navigateur,
- D'une application web ou d'un traitement exécuté sur un serveur.

### Ressource

Représente les données concernées par l'échange et exposées au moyen d'une API par l'entité qui en a la responsabilité.



### Utilisateur

Représente la personne utilisant l'application consommatrice et dont l'authentification est assurée par l'entité qui en a la responsabilité.

Certains cas d'utilisation (ex : affichage de la synthèse des comptes du client) mettent en jeu une chaîne applicative complexe, décomposables en plusieurs appels élémentaires d'API. Le cas échéant :

- Chaque appel élémentaire est ramené au triptyque Application / Ressource / Utilisateur
- Lorsqu'un échange complexe implique des appels de ressources en « cascade » la ressource appelante est considérée comme une application consommatrice de la ressource appelée.

La notion d'utilisateur peut représenter une personne physique ou un automate. Dans ce dernier cas, l'automate est considéré comme un utilisateur à part entière, distinct de l'application consommatrice, et doit donc être authentifié.

**Les entités du Groupe ou partenaires extérieurs peuvent endosser les responsabilités suivantes :**

Type d'objet	Entité responsable
<b>Application consommatrice</b>	- Toute entité du Groupe Crédit Agricole ou partenaire extérieur responsable d'une application devant consommer des API du Groupe Crédit Agricole.
<b>Ressource</b>	- Toute entité du Groupe Crédit Agricole exposant des API.

<b>Utilisateur</b>	<ul style="list-style-type: none"> <li>- Le réseau de distribution pour ses clients, collaborateurs<sup>1</sup>, prospects, prescripteurs<sup>2</sup> ou automates.</li> <li>- Le producteur pour ses clients, prescripteurs, collaborateurs ou automates.</li> <li>- Une société extérieure au Groupe pour ses collaborateurs ou automates.</li> </ul>
--------------------	---

## 2.2 Principaux cas d'usage

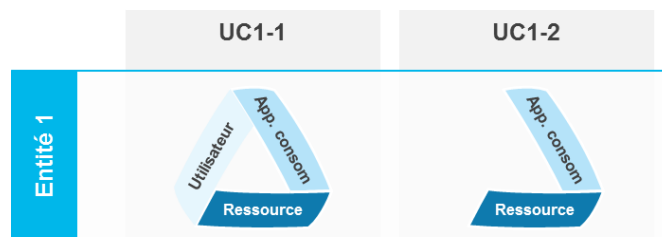
Les paragraphes ci-dessous synthétisent les différents cas d'utilisation d'une API, en fonction du nombre d'entités impliquées et de leurs rôles respectifs. La notion d'entité renvoie aux entités du Groupe Crédit Agricole ou à des partenaires extérieurs au Groupe.

Chaque cas d'usage précise :

- Les objets concernés : utilisateur, application consommatrice, ressource
- Pour chaque objet, l'entité qui en a la responsabilité

### 2.2.1 Cas d'usage impliquant 1 entité

Le schéma ci-dessous illustre les cas de consommation d'API impliquant 1 seule entité. S'agissant d'échanges internes à une entité, l'application de la norme ne revêt pas de caractère obligatoire.



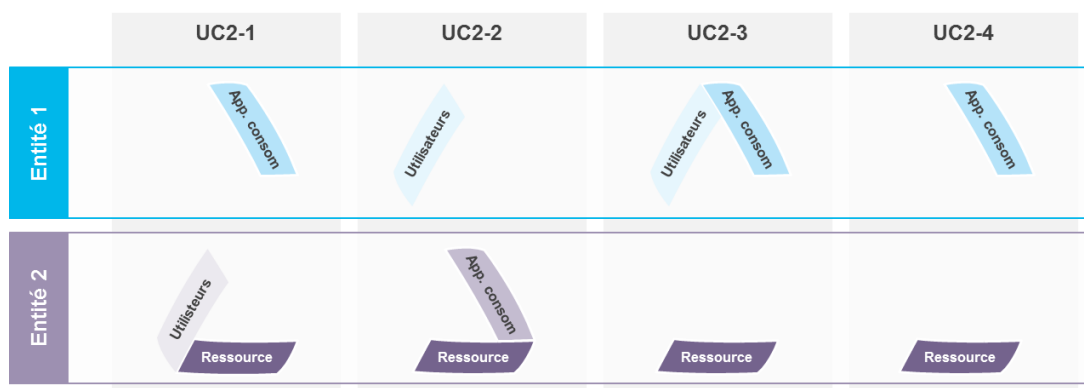
Type d'usage / Exemple	
<b>UC1-1</b>	<ul style="list-style-type: none"> <li>- Consultation du solde d'un client LCL depuis le site particulier</li> <li>- Ajout d'IBAN depuis le site NPC pour un client CRCA</li> <li>- Réalisation d'actions collaborateur depuis le PUC : <ul style="list-style-type: none"> <li>- Accéder au détail d'un compte de chèque ou d'un compte d'épargne disponible</li> <li>- Faire une demande de virement ou de chéquier</li> </ul> </li> </ul>
<b>UC1-2</b>	<ul style="list-style-type: none"> <li>- Orchestration de tâches dans l'exécution d'un processus long (BPM)</li> </ul>

### 2.2.2 Cas d'usage impliquant 2 entités

Le schéma ci-dessous illustre les cas de consommation d'API impliquant 2 entités, l'une d'elle pouvant être extérieure au Groupe.

<sup>1</sup> Ou un représentant exemple : le cadre d'une coopération, d'un partenariat type AVEM.

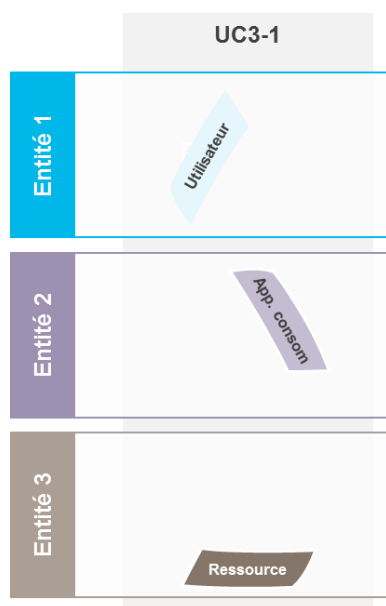
<sup>2</sup> Collaborateur d'une société externe au Groupe réalisant des ventes de produits du Groupe CA



Type d'usage / Exemple	
<b>UC2-1</b>	<ul style="list-style-type: none"> <li>- Consommation des API CA-TS par l'application producteur CA-Titres, avec délégation à CA-TS de l'authentification du client CRCA.</li> <li>- API DSP2 consommées par une entité externe (ex : Agrégateur)</li> </ul>
<b>UC2-2</b>	<ul style="list-style-type: none"> <li>- Tous les cas d'usage de consommation de ressource des clients par des applications des producteurs en contexte internet comme PrediWeb et NewSesame : le propriétaire des données "appartient" à l'entité de distribution (Caisse Régionale, LCL), la ressource et l'application consommatrice sont localisées dans l'entité producteur.</li> </ul>
<b>UC2-3</b>	<ul style="list-style-type: none"> <li>- Consommation de ressource d'un producteur depuis Ma Banque ou NPC (service synthèse) : l'utilisateur et l'application consommatrice sont localisées du côté de l'entité de distribution. Les ressources sont exposées par le producteur.</li> <li>- Consommation d'API CA-PS par CA-CF dans le cadre du projet 3XCB</li> <li>- Consommation d'API CA-TS par CA-SA dans le cadre du projet Agilor (avec authentification de l'utilisateur par CA-SA)</li> </ul>
<b>UC2-4</b>	<ul style="list-style-type: none"> <li>- Appel des services SCAD pour l'authentification forte des clients CRCA via X-Connect</li> <li>- P2P Paylib dans le cas d'un client bénéficiaire CRCA</li> </ul>

## 2.2.3 Cas d'usage impliquant 3 entités

Le schéma ci-dessous illustre les cas de consommation d'API impliquant 3 entités, deux d'entre elles pouvant être extérieures au Groupe.



Type d'usage / Exemple	
<b>UC3-1</b>	<ul style="list-style-type: none"> <li>- Application producteur qui appelle une ressource d'un autre producteur en contexte client. Le client appartient à l'entité de distribution.</li> <li>- Application Pack Auto entre CA-CF et PACIFICA : dans le cadre de l'ouverture d'un contrat crédit revolving pour l'achat d'une voiture, demande de tarif d'assurance correspondant au véhicule à PACIFICA. L'utilisateur appartient à l'entité de distribution.</li> </ul>

## 2.3 Architecture des échanges (stateless)

Les appels de services reposent sur une architecture de type REST<sup>3</sup> sur http.

**Les services sont "stateless" (sans état) : on ne gère pas, côté serveur d'application du fournisseur, un contexte de session technique destiné à mémoriser des informations en prévision d'un appel ultérieur.** Chaque requête doit contenir toute l'information nécessaire pour permettre au serveur d'application du fournisseur de comprendre la requête, sans avoir à dépendre d'un contexte conservé sur l'instance de l'application et issu de requêtes précédentes.

## 2.4 Niveau de maturité REST

REST n'est pas un modèle d'architecture en soi mais plutôt un ensemble de principes qui constituent un "style d'architecture". Ces principes peuvent être adoptés en totalité ou en partie. On parle de niveau de maturité.

Un modèle de maturité, comprenant plusieurs niveaux allant de l'adoption minimum jusqu'à l'adoption complète, a été décrit par Leonard Richardson (voir [RMM] pour plus de détails). Il permet de se positionner dans la trajectoire d'adoption et de mesurer l'effort à fournir pour progresser.

<sup>3</sup> REST : Representational State Transfert. Style d'architecture dont le Web constitue une implémentation s'appuyant sur le protocole applicatif http. Voir [REST] en annexe pour plus de détail.

Le graphique ci-dessous, inspiré d'un article de Martin Fowler<sup>2</sup>, illustre la trajectoire d'adoption possible des principes de REST.

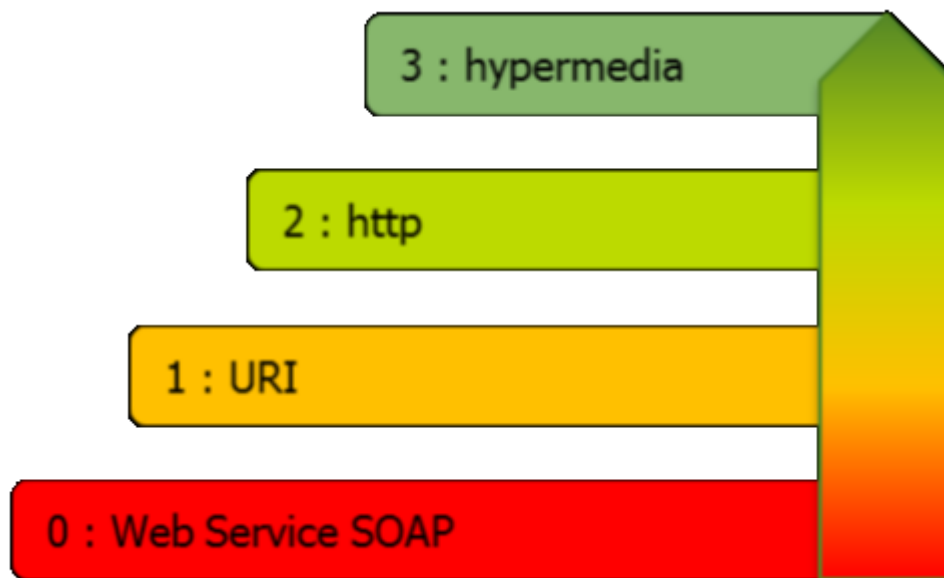


Figure 1 - Modèle de maturité de Leonard Richardson

Le tableau ci-dessous présente brièvement les différents niveaux et leurs caractéristiques :

Niveau	Description
0	HTTP est utilisé comme protocole de transport mais sans en respecter tous les principes. On identifie des services, qui ne comprennent qu'une adresse et des méthodes, et non des ressources. Les requêtes sont généralement de type POST et les données sont échangées sous forme de messages (XML ou SOAP par exemple) portés par le body. Les appels de Web Services SOAP mis en œuvre au Crédit Agricole rentrent dans ce niveau.
1	Introduction de la notion de ressource, identifiée par une URI et qui peut être appelée individuellement. On adresse donc des instances d'objets plutôt que des fonctions auxquelles on passe des arguments. Néanmoins, à ce niveau, toutes les requêtes utilisent généralement la méthode POST.
2	Utilisation standard des différents verbes http pour interagir avec les ressources. http est utilisé comme protocole applicatif dans le respect de la spécification y compris pour la négociation de contenu (utilisation des headers). Les codes retours standard sont également utilisés. La plupart des services REST exposés sur Internet mettent en œuvre ce niveau.
3	Utilisation des liens hypermédia. Les actions envisageables sur une ressource sont restituées sous forme de liens lors d'une requête sur la ressource.

**Le niveau d'adoption retenu dans le cadre des échanges est de 2 minimum :**

1. Utilisation de la notion de ressource et identification des ressources par des URI
2. Utilisation de HTTP comme protocole applicatif dans le respect des spécifications RFCs (7230-7237 et 5789) : utilisation standard des méthodes, en-têtes et codes retours y compris pour la négociation de contenu et le cache (utilisation des headers).

### 3. Utilisation possible, sans obligation, des liens hypermédia.

L'utilisation des liens hypermedia est possible si deux entités souhaitent les utiliser dans le cadre d'un échange. Dans la suite du document, les spécifications se situent au niveau 3 afin que les entités qui souhaitent l'utiliser disposent des éléments nécessaires.

### **Le transport des messages se fait en utilisant HTTP 1.1.**

Bien que la spécification de http V2 soit publiée, il est prématuré d'en généraliser l'adoption.

### 3 Définition des API

#### 3.1 Conception

**L'interface d'une API doit être décrite par deux documents mis à disposition des développeurs via un portail :**

1. Un document de spécification fonctionnelle
2. Un fichier "Swagger" qui constitue le contrat d'interface technique de l'API

**Les deux documents de description de l'API (swagger et spec. Fonctionnelle) sont de la responsabilité du fournisseur de l'API et se suffisent à eux-mêmes pour appeler l'API.**

##### 3.1.1 Spécification fonctionnelle

La documentation fonctionnelle de l'API constitue le point d'entrée des développeurs souhaitant accéder aux ressources qu'elle expose. Elle doit décrire de la manière la plus complète possible le fonctionnement de l'API : cas d'usage, acteurs, fonctionnalités et règles de gestion, diagrammes de séquence, modèle de sécurité (dont niveau d'authentification requis), modèles de données, etc.

Dans le cas d'une API exposée à des partenaires externes au Groupe, il est recommandé de réaliser une version anglaise de la documentation fonctionnelle.

La spécification produite par le STET dans le cadre de l'application de la Directive Européenne sur les Services de Paiement (DSP2) constitue un exemple complet en la matière :



API DSP2  
STET\_V1.2.3\_final.pc

(lien : [https://service.ca-mocca.com/maia-collab/rc/1/API%20DSP2%20STET\\_Documentation.pdf](https://service.ca-mocca.com/maia-collab/rc/1/API%20DSP2%20STET_Documentation.pdf))

##### 3.1.2 Fichier Swagger

**Le contrat d'interface technique de l'API est décrit au moyen de la spécification Swagger v2.0 ou supérieure. Le fichier « Swagger » est mis à disposition des développeurs via l'API Store, au format JSON. Il est de la responsabilité du fournisseur de l'API.**

Le fichier Swagger normalise :

- La version d'API et son uri de base,
- Les ressources exposées et leurs uri relatives,
- Les opérations supportées sur ces ressources,
- Les paramètres génériques à transmettre dans tous les appels (id de corrélation, infos canal...),

- Les paramètres à transmettre pour chaque cas d'usage décrit dans la spécification applicative (header, body, query),
- Les erreurs à gérer et le format des réponses,
- Les règles de sécurité (HTTPS, OAuth2 / OIDC, signature...)

**Concernant le contrat d'interface de l'API (le fichier « Swagger »), la langue anglaise est retenue pour nommer l'ensemble des ressources (et leur URI), des paramètres en entrée et des données restituées en réponse.**

Concernant la rédaction du fichier Swagger, la **tour de contrôle API** ([lien](#)) a publié :

- Un guide de bonne pratique, nommé « [best practice Swagger](#) » :



- Des modèles de fichiers swagger : [lien](#)

À titre d'exemple, ci-dessous le fichier Swagger correspondant à la spécification produite par le STET :



Lien : [https://service.ca-mocca.com/maia-collab/rc/1/StetPsd2Api\\_1\\_3\\_HAL.yaml](https://service.ca-mocca.com/maia-collab/rc/1/StetPsd2Api_1_3_HAL.yaml)

### 3.1.3 Format des uri

#### 3.1.3.1 Principes généraux

**Une URI permettant d'identifier une ressource est de la forme :**

`http(s) : //<hostName>[:<port>]/<APIName>/<APIVersion>/<resourceName>`

- **hostName** : le nom d'hôte caractérise le domaine qui héberge la ressource. Le format du nom d'hôte est contraint par des règles de nommages rappelées au chapitre suivant.
- **port** : Par défaut le port 443 est associé au protocole HTTPS.
- **APIName** : le libellé identifiant l'API. L'anglais est préconisé pour nommer une API [\[BPAPI\]](#). Il est recommandé de suivre les bonnes pratiques décrites dans le document référencé en [\[BPAPI\]](#).
- **APIVersion** : la version de l'API. les règles de gestion des versions d'API sont exposées au chapitre 3.3.2.
- **resourceName** : le chemin permettant d'identifier la ressource. C'est une chaîne de caractères au format d'un chemin de fichier (Path). L'anglais est préconisé pour nommer une Ressource [\[BPAPI\]](#). Il peut comprendre un segment identifiant un contexte technique. Il est recommandé de suivre les bonnes pratiques décrites dans le document référencé en [\[BPAPI\]](#).

Exemple :



[https://api.ca-sa.group.gca/struct\\_ref/v1/offices/ql4052](https://api.ca-sa.group.gca/struct_ref/v1/offices/ql4052)

## 3.1.3.2 Règles de nommage du nom d'hôte pour une URI Intranet

Il doit être conforme aux règles de nommage groupe, rappelées ci-dessous.

### Format du nom d'hôte

**Le nom d'hôte devra être conforme aux normes Groupe de nommage des URL Intranet (voir [NHOT](#) pour plus de détail) :**

**URL pour une application Intranet accessible niveau Groupe :**

<Environnement>-<nomapplication>.<Proprietairedelapplication>.group.gca

### Règles de nommage des environnements

<Environnement> caractérise l'environnement dans lequel est déployé le serveur ciblé. Il correspond à l'étape du cycle de développement de la fonction. On distingue les valeurs suivantes :

Étape du développement	Valeur de <Environnement>
Intégration	int ou int<i> (intégration numéro i)
Homologation	hom
Pré-production	pre
Formation	form
Certification	certi
Recette	rct

Pour l'environnement de production, la variable <Environnement> n'est pas renseignée.

### Nommage des applications

Aucune règle de nommage n'est définie.

Cependant, dans le cadre d'une URL d'API, il est autorisé de remplacer ou de préfixer avec un « - » le <nomapplication> par « api ». Le nom d'hôte devient alors :

<Environnement>-**api**.<Proprietairedelapplication>.group.gca

OU

<Environnement>-**api**-<nomapplication>.<Proprietairedelapplication>.group.gca

### Nommage du <Proprietairedelapplication>

Il s'agit d'un alias du nom de l'entreprise ou de la marque commerciale propriétaire de l'application. Voir le référentiel des noms dans l'annexe de la norme : [NURL](#)

### Autres contraintes

Les noms utilisés comprennent 12 caractères maximum.

On doit trouver au maximum 5 niveaux de sous domaine, racine comprise.

Pour le nom de l'environnement, de l'application et du propriétaire (dans le nom d'hôte), la bonne pratique du Groupe est de se limiter aux caractères alphanumériques et au « - » : **[0-9a-zA-Z-]**

Exemple 1 :

```
hom-api-myapp.entite1.group.gca
```

Exemple 2 :

```
api.entite2.group.gca
```

#### 3.1.3.3 Règle de nommage du nom d'hôte pour une URI exposée sur internet

Celui-ci est de la responsabilité de l'Entité propriétaire du nom de domaine. En conséquence, le choix du nom d'hôte doit suivre les procédures de validation prévues par les Entités concernées. Par exemple, pour « credit-agricole.fr », il est attribué par l'unité Département de l'information de la direction de la communication groupe de Crédit Agricole S.A. (CASA/DG/SGL/DCG/DI) auprès de qui une demande doit être faite.

Cependant pour assurer la cohérence des noms d'hôte d'API au sein du Groupe, il est préconisé de respecter le nommage suivant :

```
<Environnement>-api.<sous-domaine*>.<domaine de l'Entité>
```

*\* le sous-domaine est optionnel*

#### 3.1.4 Données d'en-tête

**Afin d'homogénéiser les pratiques lors de la consommation des API, plusieurs structures transmises systématiquement par les applications consommatrices dans des en-têtes HTTP ont été définies. On distingue :**

1. Les données à transmettre quel que soit le type d'échange (API privée ou Open API)
2. Les données nécessaires pour assurer le bon fonctionnement des échanges entre entités du Groupe

Les données d'en-têtes ont pour objectif d'assurer les fonctions de sécurité, routage et traçabilité des échanges par les différentes entités concernées. Elles ne doivent en aucun cas impacter le fonctionnel de la ressource appelée.

### 3.1.4.1 Cas général

Le tableau ci-dessous recense les données d'en-tête (header) à intégrer à tout appel d'API :

Nom du Header	Description
<b>Authorization</b>	Le contenu de cet header varie selon la cinématique d'authentification retenue. Son utilisation est décrite dans le chapitre 0.
<b>Correlationid</b>	<p>L'identifiant de corrélation permet d'assurer la traçabilité des actions initiées par l'application consommatrice. Cet identifiant est généré par l'application qui initie la transaction (ex : portail distributeur). Il doit être propagé tout au long de la chaîne applicative. Si cet identifiant n'est pas fourni par l'application consommatrice, il est généré par le fournisseur de la première ressource appelée dans la chaîne.</p> <p>Sa construction suit la norme Groupe : il est constitué d'un UUID V4 (nombre aléatoire), valeur sur 128 bits.</p> <p>La représentation sous forme de chaîne de caractères d'un UUID suit un format bien précis dont la longueur est 36 caractères.</p> <p>Cet identifiant est <u>obligatoire</u> dans les échanges inter entités du Groupe. Dans le cas de consommations d'API depuis l'extérieur, s'il n'est pas renseigné par l'application consommatrice, alors il devra être généré par la première ressource appelée de la chaîne applicative.</p>

### 3.1.4.2 Cas des échanges au sein du Groupe (données d'en-tête)

Les entités ont la possibilité de transmettre des données de contexte spécifiques dans l'entête des requêtes. Ces données sont généralement réservées à gérer les cas d'usage de routage et/ou de traçabilité. Si ces données doivent être sécurisées, alors elles devront être signée et/ou chiffrée tel que définie au paragraphe [5.2.3.](#)

D'une manière générale, il est préconisé de limiter l'usage de ces données d'entête spécifique et de privilégier le positionnement de ces données applicatives dans le corps de la requête.

Certaines de ces données de contexte spécifiques au Groupe ont fait l'objet d'un nommage normalisé.

#### 3.1.4.2.1 Règles de nommage générique des données d'en-tête

1/ Par soucis de simplification, les données de contexte seront positionnées à plat dans l'entête de la requête (pas de regroupement au sein d'une structure JSON)

2/ Ces données d'entête seront préfixées par « **Context-** »

3/ La convention de nommage « Train-Case » sera utilisée pour le nommage des entêtes.

## 3.1.4.2.2 Objet Context-Uom-Code

**Context-Uom-Code** est une donnée d'en-tête propres aux échanges internes au Groupe.

### Spécification

Nom du Header	O/F	Description
<b>Context-Uom-Code</b>	F	<p>Identifiant national de l'entité pour laquelle l'API est appelée (i.e. entité destinataire, dont on veut utiliser les données). Cet identifiant est notamment utilisé par les Caisses Régionales pour router les requêtes API vers la source de données (MQ, Mainframe, BDD, etc.) de la CR souhaitée.</p> <p>Identifie également l'entité pour le compte de laquelle sera émis un Compte Rendu s'il en est émis un.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Context-Uom-Code doit refléter l'uom sur laquelle l'utilisateur travaille.</b></p> </div> <p>Attention, pour rappel, cette donnée d'en-têtes doit avoir pour unique objectif d'assurer les fonctions de sécurité, routage ou de traçabilité des échanges. <b>Elles ne doivent en aucun cas impacter le fonctionnel de la ressource appelée (car la donnée n'est pas sécurisée).</b></p> <p>Chaîne de 5 caractères alphanumériques cadrés à gauche, renseignée par l'application selon la nomenclature des <a href="#">UO Métier CAM0303</a>.</p>

## 3.1.4.2.3 Objet Context-Initial

**Context-Initial** est une donnée d'en-tête propres aux échanges interne au Groupe.

### Spécification

Nom du Header	O/F	Description
<b>Context-Initial</b>	F	<p>Donnée caractérisant chez l'entité à l'initiative d'un échange le contexte d'exécution à partir duquel l'échange intervient.</p> <p>L'échange initial peut être le lancement d'une IHM (à l'initiative d'un distributeur en général), ou un appel de service.</p> <p>Exemple de cas d'utilisation : le distributeur lance une IHM producteur, puis, depuis l'IHM du producteur en cours de lancement on appelle un service chez le distributeur. L'entité à l'initiative de l'échange est alors le distributeur.</p> <p>Cette donnée est nécessaire lorsque :</p> <ul style="list-style-type: none"> <li>- Pour un projet, une succession d'échanges circulaires requiert que l'on connaisse l'environnement de départ.</li> <li>- Les environnements de tests entre producteurs et distributeurs ne sont pas alignés (par exemple plusieurs environnements du distributeur adressent le même environnement chez le producteur).</li> </ul> <p>Elle n'est donc à utiliser que pendant les phases projets précédant la production (en production les environnements sont alignés) et seulement lorsque le projet le justifie (appels circulaires).</p>

	<p>Dans le cas d'une utilisation pour le lancement d'IHM puis l'appel d'API :</p> <ul style="list-style-type: none"> <li>- Elle doit être renseignée dans le contexte d'IHM lors du lancement de l'IHM.</li> <li>- Elle doit ensuite, lors de l'appel du service en rebond, être reportée dans la présente donnée.</li> </ul> <p>Une fonction de routage mise en œuvre chez l'entité à l'initiative de l'échange (le distributeur pour le cas d'utilisation considéré), l'utilisera pour diriger l'appel du service vers l'environnement à partir duquel a été émis le lancement d'IHM.</p> <p>Chaîne de 5 caractères alphanumériques maximum.</p>
--	--

### 3.2 Publication et exposition

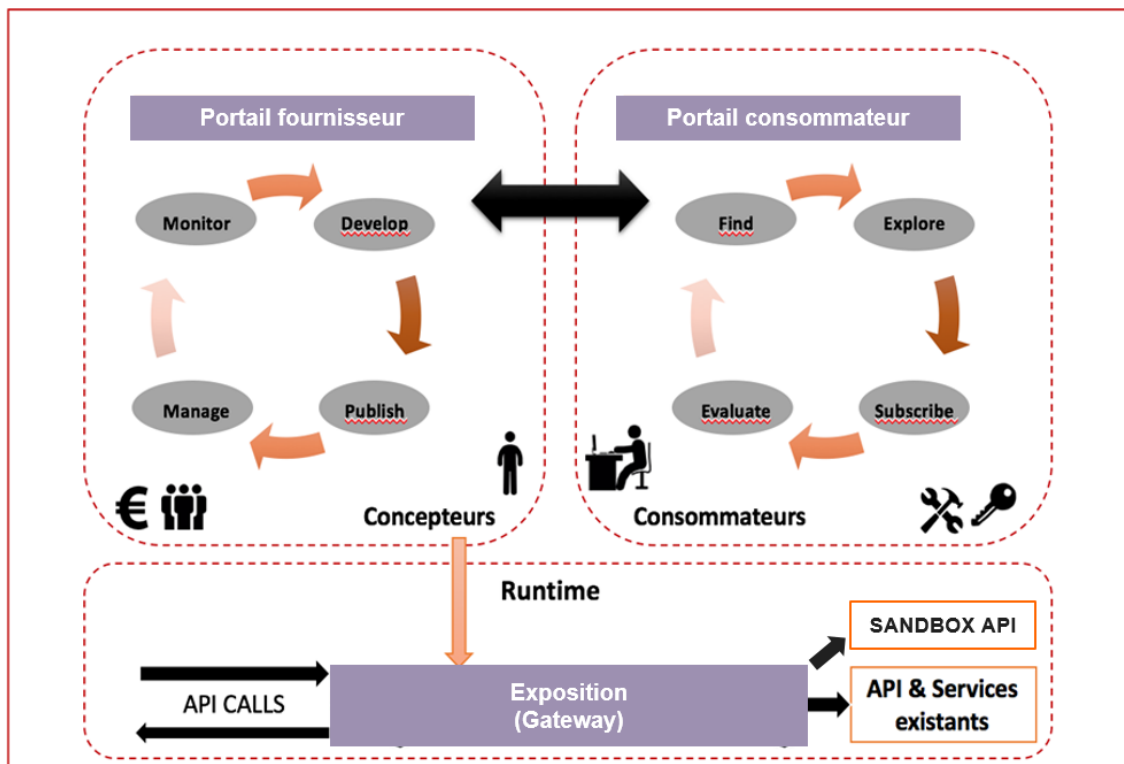
L'animation de la relation Consommateur / Fournisseur dans une démarche d'API repose sur la mise à disposition de fonctions facilitant le développement, la publication et l'utilisation des API exposées par une entité. Ces fonctions sont regroupées sous le terme « Gestion des API » (ou API Management).

#### 3.2.1 Gestion des API (API Management)

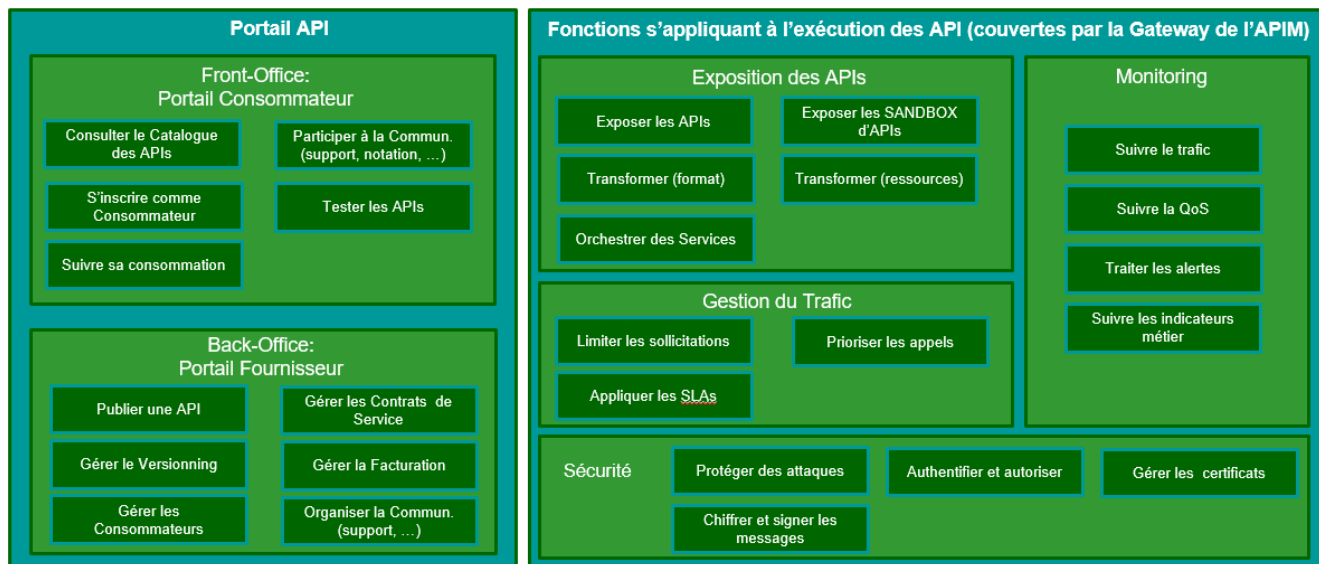
La gestion des API regroupe 3 grandes fonctionnalités :

- **Portail fournisseur (accessible aux développeurs/fournisseurs d'API)**
  - Permet aux fournisseurs d'API de publier facilement leurs API, partager la documentation, et recueillir les commentaires des consommateurs
  - Permet de gouverner le cycle de vie de celle-ci depuis la création de l'API jusqu'à son dé-commissionnement
  - Permet de gérer les consommateurs (validation des demandes d'accès à une API, etc.)
  - Expose des Dashboard d'usage de l'API à destination du fournisseur
- **Portail consommateur (accessible aux consommateurs d'API)**
  - Met à disposition le catalogue d'API disponibles pour les consommateurs
  - Héberge la documentation technique, fonctionnelle et les usages de chaque API (tags, description)
  - Permet d'enrôler les applications consommatrices et de souscrire aux API
- **Exposition (fonction pouvant prise en charge par une Gateway)**
  - Permet d'exposer des API
  - Sécurisation des API (Authent. des applications, contrôle des autorisations d'accès à l'API, vérification des jetons OAuth/OIDC)
  - Monitoring du trafic (remontée des données vers la supervision)
  - Gestion de la qualité de service (tamponnage des flux ou Throttling)
  - Gestion du cache
  - Permet d'exposer une SANDBOX (bac à sable) des API

Le schéma ci-dessous illustre les fonctions précédemment décrites et leur positionnement respectif dans la chaîne d'utilisation d'une API.



Les fonctionnalités précédemment décrites ont également été présentées lors du GT APIisation des SI de l'AEG, sous la forme suivante :



Chaque entité fournisseur d'API doit mettre en œuvre les fonctions d'API Management en adéquation avec la mise en œuvre de la démarche API dans son contexte. L'utilisation d'un outil d'API Management doit être considérée comme un facilitateur (pas une obligation) à la mise en œuvre de ces fonctions. Si un outil d'APIM est utilisé, la solution de référence retenue par le Groupe ([lien](#)) devra être choisit (sauf si justification). Si aucun outil n'est utilisé, les fonctions d'API

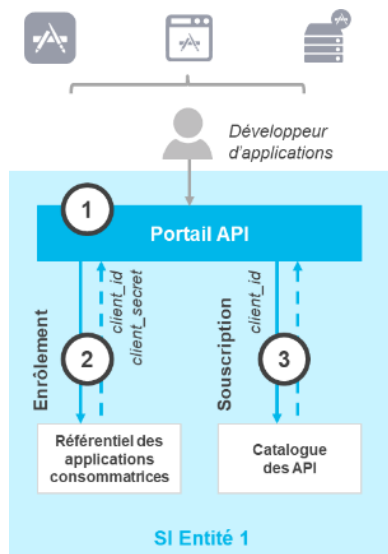
**Management nécessaires devront être mises en œuvre d'une autre manière (développements applicatifs, etc.)**

### 3.2.2 Gestion des consommateurs

Le processus de gestion des consommateurs a pour objectif de limiter l'accès aux API d'une entité aux applications consommatrices dûment autorisées par celle-ci. On distingue trois phases :

- **Accès au portail consommateur d'API :** toute personne (développeur d'application consommatrice ou mandataire) souhaitant consommer les API d'une entité doit être déclarée auprès de celle-ci et habilitée à accéder à son portail API. Les moyens et processus sous-jacents sont du ressort de chaque entité fournisseur d'API.
  
- **Enrôlement d'une application consommatrice :** cette étape permet au développeur d'application ou à son mandataire de déclarer l'application consommatrice sous sa responsabilité. Il obtient en retour un identifiant d'application (nommé « client\_id » par le standard OAuth2) et un mot de passe (nommé « client\_secret » par le standard OAuth2) lui permettant d'authentifier l'application consommatrice auprès des fournisseurs d'API. Ces authentifiants sont propres à l'application consommatrice et ne doivent pas être divulgués.  
L'enrôlement permet d'acquérir des informations complémentaires sur l'application consommatrice pouvant impacter les cinématiques d'authentification (cinématiques autorisées, types de jetons délivrés, durée de validité...). Le niveau de sécurisation de cette phase d'enrôlement est du ressort de l'entité en fonction des cas d'usage à couvrir, en particulier leur criticité.  
L'exactitude des données déclarées lors de l'enrôlement sont de la responsabilité de l'entité de l'application consommatrice habilités à l'enrôlement, mais aussi, de la responsabilité de l'entité fournissant l'API qui doit valider la demande d'enrôlement si et seulement si les informations déclarées sont correctes.
  
- **Souscription à une API :** cette étape permet d'accorder le droit d'accès à une API par une application consommatrice préalablement enrôlée et identifiée par son identifiant (client\_id). La demande de souscription à une API peut donner lieu à un workflow de validation avant de l'accepter.

Le schéma ci-dessous illustre les 3 étapes précédemment décrites :



- 1 Le développeur d'application ou son mandataire accède au portail API de l'entité 1.
- 2 Le développeur d'application ou son mandataire enrôle une application consommatrice auprès de l'entité 1 et obtient en retour ses authentifiants
- 3 Le développeur d'application ou son mandataire souscrit à une ou plusieurs API pour l'application concernée

**Un fournisseur d'API doit s'assurer de la mise à disposition d'un portail consommateur pour l'enrôlement des applications consommatrices et la souscription aux API disponibles.** L'enrôlement d'une application consommatrice lui permet d'obtenir les informations à utiliser (client\_id et client\_secret) pour s'authentifier lors d'une demande d'accès à une API.

Les authentifiants délivrés à une application consommatrice sont les suivants :

- Un identifiant d'application (client\_id – SHA-256 d'une chaîne de caractères composée d'éléments garantissant l'unicité)
- Un mot de passe (client\_secret – chaîne de caractères de type mot de passe conforme aux standards de sécurité du Groupe)
- De façon optionnelle, un autre moyen d'authentification tel qu'un certificat client

### 3.2.3 Mise à disposition d'une Sandbox

Dans le cadre d'exposition d'API à des consommateurs, **il est préconisé de mettre à disposition une Sandbox** pour permettre à ces consommateurs de tester l'appel de l'API indépendamment des flux et des données de production.

Cette Sandbox aura les caractéristiques suivantes :

- Pour les API accessibles sur Internet, la Sandbox devra être disponible sur Internet aux applications enrôlées et autorisées à l'API.
- Pour les API accessibles uniquement sur l'Intranet, la Sandbox devra être disponible uniquement sur l'Intranet, aux applications enrôlées et autorisées à l'API.
- La ou les Sandbox seront basées sur des jeux de données de test fictives et anonymes, pour limiter les risques sécurité
- La ou les Sandbox seront installées sur des environnements de production isolées pour ne pas impacter les API de production



- La ou les Sandbox devront prévoir les différents mécanismes sécurisant l'accès à l'API (ident./authent. d'utilisateurs fictifs, jetons, contrôle d'accès de l'application à l'API, etc.) pour qu'ils puissent être testés.

2 niveaux de préconisations suivant le niveau d'exposition de l'API :

**Pour une API accessible à des partenaires sur Internet, il est fortement préconisé de mettre à disposition une Sandbox.**

**Pour une API Intranet ou une API Internet privée, la mise en place d'une Sandbox est une bonne pratique.**

### 3.2.4 Portail API fédérateur Groupe

Pour permettre la visibilité des API à l'échelle du Groupe (entre entités), la tour de contrôle API met à disposition un portail API fédérateur Groupe : <https://api.group.gca/>

Il est également accessible sur l'Intranet AEG : <https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/Pages/PortailAPI.aspx>

Et un [Guide du portail API fédérateur](#) est aussi disponible.

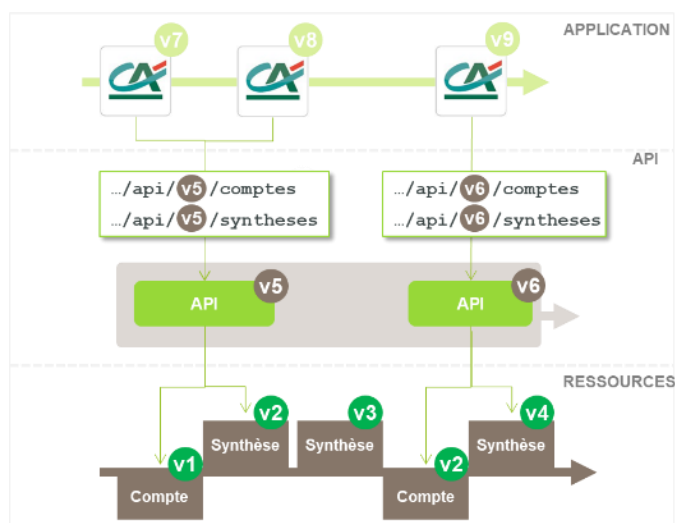
## 3.3 Cycle de vie

### 3.3.1 Généralités

**Les applications consommatrices, les API et les ressources ont des cycles de vie distincts, et donc des numéros de versions distincts.**

Le schéma ci-dessous illustre les modalités de gestion des versions :

- Il est ainsi possible de livrer des nouvelles versions des applications consommatrices sans relivrer les APIs qu'elles utilisent (dans l'exemple ci-dessous passage de l'application v1 à la v2 sans relivrer l'API).
- Plusieurs versions d'une même API peuvent être exposées en parallèle (dans l'exemple v5 et v6).
- Vu des applications consommatrices, seul le numéro de version majeur de l'API est connu (v5/v6 dans l'exemple) : dit autrement, le numéro de version des ressources n'est connu que du fournisseur de l'API.



### 3.3.2 Montée de version

Les montées de version des API sont intimement liées à celles des ressources qui les composent. Les paragraphes suivants précisent les adhérences entre leurs cycles de vie respectifs.

#### 3.3.2.1 Cas d'une ressource

Le schéma ci-dessous illustre les modalités de montée de version de la ressource :

- Si les modifications apportées au contrat d'interface permettent aux clients actuels de continuer à consommer la ressource, la montée de version n'est pas nécessaire.
- De même, un changement d'implémentation de la ressource peut être considéré comme rétro-compatible (ou non) par le fournisseur.
- Cette souplesse vise à minimiser le nombre de versions, et donc à **stabiliser les API**.
- Les consommateurs d'API doivent supporter ces mécanismes d'extensibilité, notamment **permettre que leur code continue à fonctionner en cas d'ajout d'un attribut** (ou champ) dans les réponses renvoyées par les API.

<p>Le <b>contrat d'interface</b> de la ressource change</p> <p>→ <b>Pas de montée de version systématique</b></p>		<p>L'<b>implémentation</b> de la ressource change</p> <p>→ <b>Pas de montée de version systématique</b></p>
<p><b>Exemple de modification rétrocompatible :</b></p> <p>- ajout du champ <b>devise</b> dans la réponse à la requête GET sur la ressource /operations</p> <pre> {   operation   - libelle   - montant   - type   - devise <b>NEW</b> }</pre>	<p><b>Exemple de modification non rétrocompatible :</b></p> <p>- L'opération GET sur la ressource /operations ne supporte plus le filtre sur le type d'opération</p> <pre>GET /operations?type=...</pre>	<p><b>Exemple de modification rétrocompatible</b></p> <p>- Ajout d'un mécanisme de cache à la ressource /operations afin d'améliorer ses temps de réponse</p>
<b>Pas de montée de version requise</b>	<b>Montée de version nécessaire</b>	<b>Pas de montée de version requise</b>

La **montée de version majeure** d'une ressource est décidée par le fournisseur de la ressource. Elle n'est nécessaire qu'en cas de modification non-rétrocompatible

Le tableau ci-dessous précise la caractère rétro-compatible ou non d'une modification :

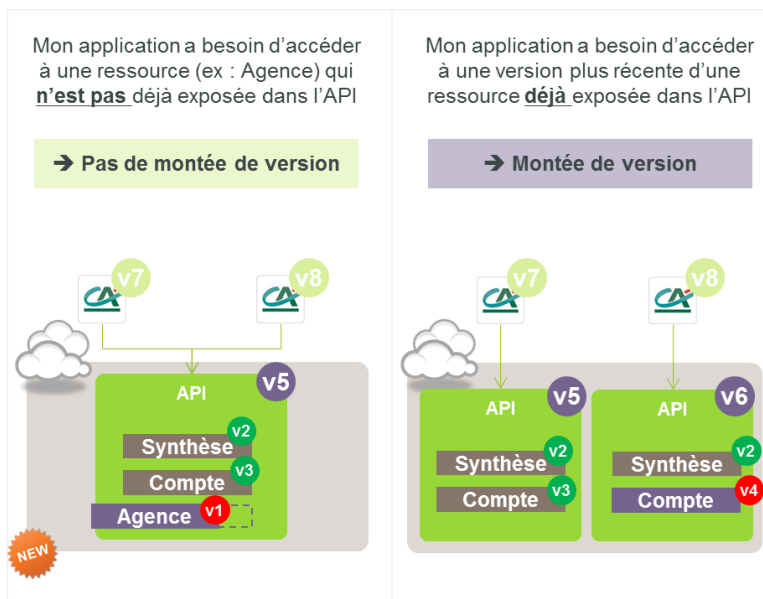
Type d'évolution	Évolution non-rétrocompatible de l'interface et/ou des règles de gestion	Évolution rétro-compatible de l'interface et/ou des règles de gestion	Correctif rétro-compatible
Type de montée de version	Montée de version majeure (X)	Montée de version mineure (Y)	Correctif (Z)

### 3.3.2.2 Cas d'une API

La **montée de version de l'API** est nécessaire uniquement à l'occasion de la **montée de version majeure** d'une ressource **déjà** exposée sur l'API

Le schéma ci-dessous illustre les modalités de montée de version de l'API :

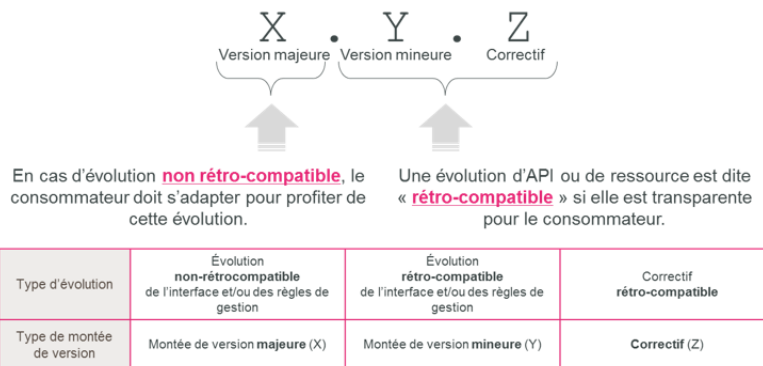
- L'ajout d'une ressource à l'API existante ne nécessite pas de montée de version : en effet, ce changement est **sans impact pour l'ensemble des consommateurs existants de l'API**.
- À contrario, l'exposition d'une version plus récente d'une ressource déjà exposée nécessite une montée de version **afin de ne pas impacter les consommateurs actuels**.



### 3.3.3 Nomenclature

La version des API est exprimé sur 1 nombre entier positif

Pour les ressources, la bonne pratique (non obligatoire) est d'exprimer la version sur 3 nombres entiers positifs (Cf. schéma ci-dessous)



### 3.3.4 Gouvernance

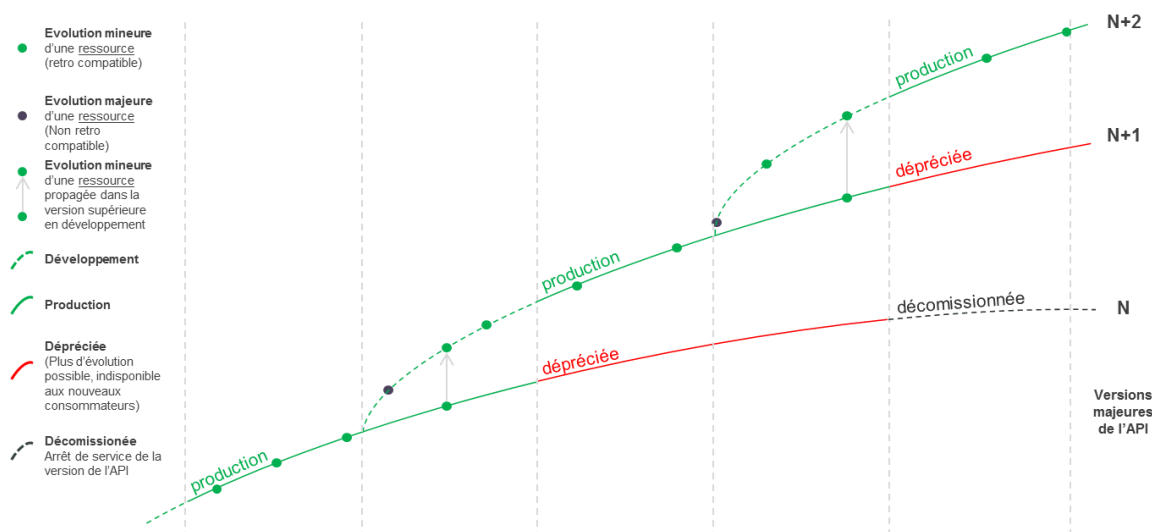
Afin d'assurer une cohérence d'ensemble des API exposées au sein du Groupe, il est nécessaire de définir des règles de gouvernance permettant d'organiser au mieux la montée de version des API, et anticiper l'impact sur les consommateurs. Une API peut être qualifiée de :

- **Nominale** : version en production, dont l'usage est à privilégier
- **Dépréciée** : version en production, dont l'usage est autorisé de façon transitoire
- **Décomissionnée** : version non accessible en production

**Il est préconisé de n'avoir que 2 versions majeures d'une API, simultanément, en production (1 nominale et 1 dépréciée)**

**La montée de version d'une API impacte l'ensemble des applications consommant l'API. Pour cette raison, toute montée de version doit rester de l'ordre de l'exceptionnel**

Le schéma ci-dessous illustre les états du cycle de vie de l'API et les transitions associées.



Le cycle de vie de l'API est rythmé par ses évolutions majeures. Le fournisseur d'API doit anticiper les montées de version et organiser les changements en coordination avec les consommateurs de l'API.

Les modalités opérationnelles du processus de gestion du cycle de vie doivent intégrer les jalons suivants :

- **Dès prise de connaissance d'un besoin de montée de version d'une API**
  - Notifier chaque application consommatrice de l'API qu'une nouvelle version est en cours de conception
  - Cette 1ère notification permet aux applications consommatrices d'avoir de la visibilité sur le cycle de vie de l'API et, finalement, d'anticiper bien en amont la dépréciation de la version en cours
- **Développement d'une version N+1**
  - La version N+1 de l'API doit intégrer
    - Les **ressources créées** dans le cadre de la version N+1
    - L'ensemble des **ressources non dépréciées** de la version N (Version mineure la plus récente)
  - Notifier chaque consommateur qu'il doit inscrire dans sa roadmap la consommation de la nouvelle version de l'API
- **Mise en production de la version N+1**
  - La mise en production de la version N+1 entraine automatiquement la **dépréciation** de la version **N**

- Toute nouvelle application consommatrice de l'API doit consommer la version N+1
- Toutes les évolutions mineures de ressource doivent être déployées dans la version N+1
- Dans le portail consommateur, le statut de l'API passe de version N à dépréciée
- **Développement d'une version N+2**
  - La nouvelle version de l'API doit intégrer :
    - Les **nouvelles ressources** créées dans le cadre de la version N+2
    - L'ensemble des **ressources non dépréciées** de la version **N+1** (Version mineure la plus récente)
  - Notifier chaque consommateur qu'il doit inscrire dans sa roadmap la consommation de la nouvelle version de l'API
- **Mise en production de la version N+2**
  - La mise en production de la version N+2 entraîne automatiquement la **dépréciation** de la version **N+1**
  - La mise en production de la version N+2 implique le déclenchement du **décommissionnement** de la version **N**
  - Toute nouvelle application consommatrice de l'API doit consommer la version N+2
  - Toutes les évolutions mineures de ressource doivent être déployées dans la version N+2
  - Dans le portail consommateur, le statut de l'API passe de version N+1 à dépréciée

Quelques préconisations et bonnes pratiques supplémentaires, pour la gestion du cycle de vie :

Gestion d'un API dépréciée par les consommateurs :

- Concernant les délais accordés aux consommateurs d'une API dépréciée pour migrer, il sera établi en concertation entre le fournisseur de l'API et ses consommateurs, en fonction des contraintes de chacun.
- Les consommateurs d'une API dépréciées ont la responsabilité de gérer les obsolescences et donc de migrer vers la nouvelle API (dans un délai raisonnable à convenir avec le fournisseur).

Gestion d'un API dépréciée par le fournisseur de l'API :

- Il est préconisé de ne pas utiliser un code retour HTTP spécifique pour signifier aux consommateurs qu'une API est dépréciée (exemple : code retour 410). En effet, ce changement de code retour HTTP risque d'être mal interprété par les consommateurs et de créer une erreur.
- En cas d'API dépréciée, il est une bonne pratique que de tracer les consommateurs qui utilisent encore l'API et de les relancer régulièrement pour qu'ils migrent.

### 4 Définition des ressources

#### 4.1 Généralités

Une ressource peut être tout ce que l'on considère comme suffisamment important pour être géré et représenté en tant que tel.

Pour y faire référence on utilise une représentation de la ressource, qui correspond à l'état courant de la ressource, caractérisé par la valeur de ses différents attributs.

La ressource est décrite par :

- Une URI d'identification (l'anglais est préconisé)
- Des métadonnées spécifiques à la représentation de la ressource :
  - Le type de représentation exprimé au moyen du Type MIME (application/json, application/pdf, text/xml, text/html, image/jpeg, video/mpeg, ...),
  - La date de dernière modification,
  - Les données nécessaires au cache.
- La valeur des différents attributs de la ressource :
  - Son identifiant,
  - Les autres attributs.

#### 4.2 Conception des ressources

Se référer au document référencé en [[BPAPI](#)].

Concernant les codes erreurs fonctionnels, leur définition est de la responsabilité de l'entité fournisseur de l'API.

## 5 Sécurisation des échanges

### 5.1 Exigences générales

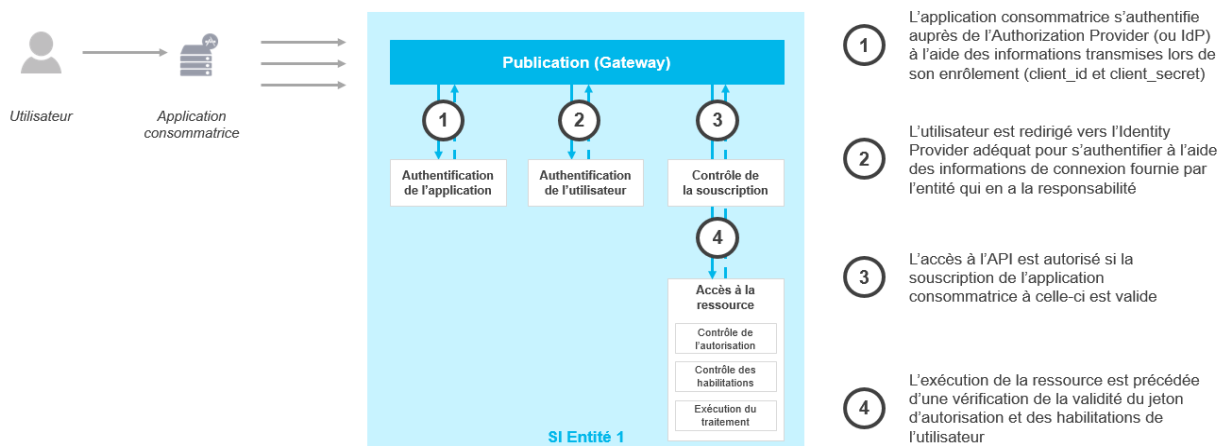
Les résultats et exigences des différents travaux de sécurité du Groupe (CASA/SRI) devront s'appliquer aux API au même titre que pour les autres développements applicatifs (et notamment le [référentiel SECAPI](#) pour le développement, les politiques de sécurité poste de travail, mobiles et serveurs, la politique de sécurité des DataCenter, etc.). Ces règles de sécurité générales ne figurent donc pas dans ce document. Seules sont ajoutées les exigences spécifiques à la mise en œuvre des API dans les échanges.

Pour définir les exigences sécurité spécifiques à la mise en œuvre des API REST, la réalisation d'une analyse de risque MESARI [cf. [ARAPI](#)] a été déléguée au GT SECAPI, piloté par Crédit Agricole S.A. SIG/SRI. Cette analyse a permis de constituer un catalogue de mesures de sécurité à appliquer et permettant de répondre à différents contextes (application s'exécutant dans un navigateur ou application native, dispositif maîtrisé ou non) et différents niveaux d'exigences (DICP).

La sécurisation des API recouvre 3 thèmes :

- L'application d'exigences générales de sécurité relatives à l'exposition de services et données
- L'authentification et autorisation de l'application consommatrice et de l'utilisateur
- La gestion des habilitations fonctionnelles de l'utilisateur

Le schéma ci-dessous illustre le principe général fonctionnement :



**Les fournisseurs d'API doivent garantir le respect des règles de sécurité définies dans le présent document et celles définies comme des standards du Groupe.**

L'Analyse de Risques, la conception et l'implémentation de ces API doivent intégrer ces éléments de sécurité By-Design sans préjuger des dispositions prises par les applications consommatrices, en particulier sur la gestion des permissions et des pistes d'audit.



## 5.1.1 Authentification de l'application consommatrice

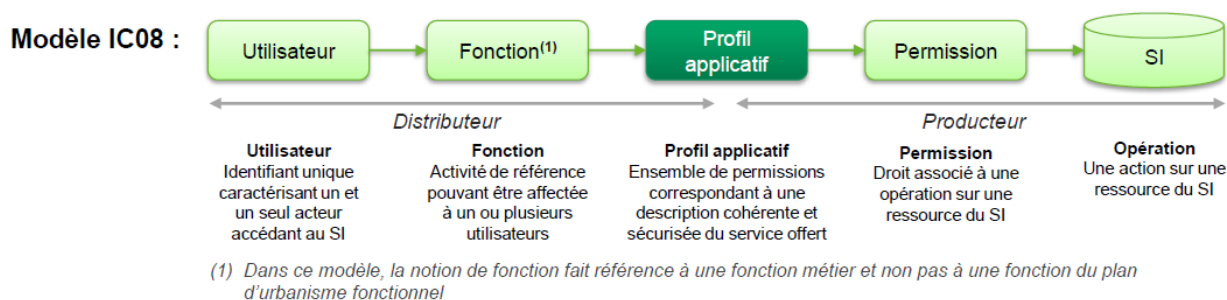
- La **consommation d'API** d'une entité nécessite un **enrôlement** préalable de l'application consommatrice auprès de celle-ci ainsi qu'une **souscription** à l'API concernée.
- Pour accéder aux API d'une entité, l'**application consommatrice doit s'authentifier** auprès de celle-ci, à l'aide des authentifiants délivrés lors de l'enrôlement préalable.

## 5.1.2 Authentification de l'utilisateur (selon cas d'usage)

- Pour accéder aux API d'une entité nécessitant **une autorisation utilisateur**, l'application consommatrice doit authentifier son utilisateur et obtenir son consentement pour accéder à ses données.
  - Conformément aux principes issus des travaux Identités et Habilitations (voir [IDHAB] pour plus de détails), l'authentification est réalisée par l'entité ayant la responsabilité de l'utilisateur, en s'appuyant sur les services mis à disposition par celle-ci.
  - Le recueil du consentement peut s'effectuer de façon **explicite** (interaction avec l'utilisateur) ou **implicite** (sans interaction avec l'utilisateur) lorsqu'il est obtenu par un autre moyen (ex : signature d'une convention de compte pour un client, signature d'une charte informatique pour un collaborateur, etc.)

## 5.1.3 Habilitations fonctionnelles de l'utilisateur

Conformément aux principes issus des travaux Identités et Habilitations (voir [IDHAB] pour plus de détails) et en application du modèle IC08, les données matérialisant les **habilitations** des utilisateurs sont gérées par l'entité ayant délégation de gestion des identités et des habilitations en correspondance avec les conventions passées avec les fournisseurs d'API. Ces habilitations sont ensuite contrôlées chez les fournisseurs d'API lors de l'**exécution des ressources**. Le profil d'habilitation applicatif est le pivot de correspondance entre les habilitations attribuées à l'utilisateur (côté Entité gérant l'utilisateur) et les permissions d'accès aux ressources (côté fournisseur d'API)



Le profil d'habilitation et les données complémentaires d'habilitation (habilitations fines) seront transmises à l'API avec les données applicatives/fonctionnelles, donc, en dehors de l'Access Token.

Voir [§5.1.6](#) pour le format de ces données d'habilitation dans le jeton JWT applicatif.

### 5.1.4 Contrôle de la souscription à l'API

- Le fournisseur d'API contrôle la **souscription** de l'application consommatrice à l'API
- L'application consommatrice présente à l'API le **jeton d'autorisation** (« access token ») de l'utilisateur aux ressources appelées, qui contrôlent la validité de ce dernier.
- Les fournisseurs de ressources peuvent juger nécessaire de s'assurer que le **niveau de confiance dans l'authentification de l'utilisateur** est en adéquation avec le risque représenté par une utilisation frauduleuse. L'application consommatrice doit disposer des données nécessaires à ces vérifications et les transmettre lors de la consommation d'une ressource.

### 5.1.5 Contrôle de l'accès à la ressource

Lorsque l'API et la ressource sont hébergées sur des instances de serveur distinctes, il est de la responsabilité de l'entité proposant l'API de sécuriser ce lien entre l'API et la ressource associée, en respectant les exigences générales de sécurité du Groupe (Cf. introduction du § 5.1).

La présente norme API ne normalisera pas ce point qui ne concerne pas les échanges inter-entités ou vers des partenaires externes.

Cependant, pour permettre la convergence des pratiques des Entités, le CESI a souhaité produire quelques préconisations et exemples de solutions concernant la sécurisation de l'accès à la ressource depuis une Gateway APIM (si utilisé) :

- Il est préconisé de contrôler l'accès à la ressource pour que l'accès à la ressource ne soit possible que depuis la Gateway pour ne pas court-circuiter les contrôles effectués sur la Gateway
- Il est préconisé que les données de sécurité présentes dans l'Access Token et devant être transmises à la ressource soient signées et/ou chiffrées. Cela assure l'intégrité de ces données et constitue une preuve du contrôle de l'Access Token par la Gateway
- Si une Gateway est utilisée pour effectuer des contrôles de sécurité préalable à l'accès à une ressource, alors, il est préconisé que cette Gateway soit positionnée « à proximité » (en terme technique et en terme de responsabilité) de la ressource. Sinon, la Gateway perd de son intérêt en terme de sécurité et les contrôles doivent être renouvelés au niveau de la ressource.

Exemples de solutions (à choisir en fonction, notamment de l'analyse de risque MESARI) pour sécuriser les échanges entre Gateway et ressources :

- Basic authentication + TLS Serveur
- Tunnel TLS entre Gateway et Ressource (certif TLS Client + certif TLS Serveur)

- Jeton JWT + Basic authentication + TLS Serveur
- Sécurisation applicative spécifique à l'entité
  - Jeton (signature et/ou chiffrement) applicatif sous un format interne à l'entité
  - Chiffrement avec une clé symétrique
  - Etc.
- Etc.

Par défaut, l'API Gateway de référence de Groupe (WSO2) mettra à disposition des API la sécurisation de l'accès à la ressource via un jeton JWT contenant les données reçues ou associées à l'Access Token (le JWT sera alors transféré via le HEADER « X-JWT-Assertion »).

### 5.1.6 Autres données applicatives : application des exigences générales de sécurité (SEC-API)

En application des standards du marché OAuth et OpenID Connect, **les données fonctionnelles (i.e. applicatives) que l'application souhaitera envoyée à l'API ne pourront pas être véhiculées dans le jeton d'autorisation (Access Token) envoyé à l'API pour en contrôler l'accès.**

Ces données fonctionnelles sont plutôt destinées à être **transmises dans le BODY** (ou dans l'URI ou la QueryString pour les requêtes GET, en prenant garde à leur limite de taille) de la requête, **indépendamment de l'Access Token**. Cela permettra d'éviter tout couplage entre les données applicatives et la solution de sécurisation des API.

En application des exigences SEC-API, **si ces données fonctionnelles sont jugées sensibles** (dans le MESARI correspondant), **elles devront être sécurisées**. C'est-à-dire signées et éventuellement chiffrées conformément aux mesures de sécurité rappelées dans les paragraphes [5.2.2.6](#) pour la signature et [5.2.3](#) pour le chiffrement. C'est l'application qui réalisera ces actions de sécurisation (pas l'Authorization Server en charge de produire les jetons de sécurité).

#### 5.1.6.1 Données d'habilitation de l'utilisateur

Parmi les données fonctionnelles/applicatives (décrites dans le paragraphe précédent), il pourra y avoir les données d'habilitation de l'utilisateur décrites au [§5.1.3](#).

Ces données d'habilitation sont à considérer comme sensibles et devront, au minimum, être signées (JWT signé).

Si ces données d'habilitation sont transmises à l'API, leur format sera le suivant :

Intitulé	O/F	Usage	Description	Format
user_entitlements	F		Sous-structure JSON regroupant les données d'habilitation de l'utilisateur	Tableau de [JSON Object]
uom_code	F	Habilitations	Code de l'UO Métier pour laquelle l'utilisateur travaille.	String
user_profile	F	Habilitations	Profil d'habilitation de l'utilisateur pour l'uom considérée	String

additional_element	F	Habilitations	Sous-structure des données complémentaires d'habilitation	Tableau [JSON Object]
name	F	Habilitations	Clé	String
value	F	Habilitations	Valeur	String

Exemple JSON des données d'habilitation :

```
{
  "user_entitlements": [{
    "uom_code" : "--",
    "user_profile" : "--",
    "additional_element": [{
      "name" : "--",
      "value" : "--"
    }]
  }]
}
```

## 5.2 Mise en œuvre des mesures de sécurité

Les mesures nécessaires pour satisfaire les exigences sont mises en œuvre pour partie dans les infrastructures (par exemple connexion TLS, antivirus, ...) et pour partie dans les applications, côté serveur et côté client ainsi que dans les messages transmis.

Ne sont présentées ici que les mesures mises en œuvre dans les applications ou API.

### 5.2.1 Bonnes pratiques de développement

Afin de protéger le navigateur et les applications Web (et API) qui l'utilisent, principalement contre les attaques de type Cross-site Scripting (XSS) plusieurs mesures doivent être mises en œuvre de manière obligatoire pour les applications Web (et non pour les applications natives qui n'utilisent pas le navigateur)

#### 5.2.1.1 Limitation des contenus accessibles aux navigateurs

Cette mesure a pour but de limiter les domaines à partir desquels un navigateur, lorsqu'il construit une page, peut télécharger des contenus susceptibles d'être infectés (scripts, images, polices, ...).

Pour ce faire :

**Le header Content-Security-Policy doit être mis en œuvre sur les serveurs applicatifs qui hébergent les applications Web (et les API).**

Il permet de définir une liste blanche des sources de contenus de confiance. **CSP fonctionne sur la base d'une page, ainsi le header doit être inclus dans chacun des flux transmis au navigateur pour construire une page** (voir [CSP](#) pour plus de détails).

Le paramétrage minimum (le plus exclusif, qui limite les possibilités de téléchargement au domaine de la page) est le suivant :

```
Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self'; upgrade-insecure-requests
```

Il permet de :

- `default-src` : filtrer les sources de différentes ressources accédées par la page comme les images, scripts, frames, polices, feuilles de style, ...
- `script-src` : filtrer les sources des scripts téléchargés par la page. Cette directive prend le pas sur la directive `default-src` ;
- `img-src` : filtrer les sources des images téléchargées par la page. Cette directive prend le pas sur la directive `default-src` ;
- `upgrade-insecure-requests` : forcer le navigateur, avant l'envoi d'une requête, à corriger les url en "http:" pour utiliser "https:" ;

Ce paramétrage minimum pourra être contextuellement élargi si l'on souhaite ajouter d'autres sources de confiance à partir desquelles le navigateur pourra télécharger des contenus ou filtrer d'autres types de contenus.

CSP permet également d'évaluer, dans un premier temps, l'impact de l'utilisation de l'en-tête `Content-Security-Policy` en utilisant l'en-tête `Content-Security-Policy-Report-Only`, avec le même paramétrage. Le navigateur signale alors les violations de sécurité mais sans blocage.

#### **5.2.1.2 Limitation de l'usage des wildcard dans l'entête CORS "Access-Control-Allow-Origin » et contrôle de l'en-tête Origin.**

Pour améliorer l'ergonomie des applications Web, il est fréquent d'émettre des requêtes asynchrones, vers des API, en tâche de fond, certaines vers des domaines différents de celui de l'application. Nativement, ces requêtes sont bloquées par le navigateur. Pour contourner cette limitation d'une manière présentant des garanties de robustesse, le W3C a spécifié le mécanisme Cross Origin Ressource Sharing (CORS - Voir [\[CORS\]](#) pour plus de détails).

Toutefois, ce dispositif, s'il est utilisé sans restriction, peut exposer les serveurs à des menaces. Pour les réduire, lorsqu'un navigateur émet une requête vers une ressource qui n'est pas publique, le serveur qui l'héberge doit vérifier que l'émetteur est digne de confiance. Pour ce faire, il récupère la valeur du domaine de l'émetteur dans l'en-tête `Origin` de la requête, renseigné automatiquement par le navigateur, et vérifie son existence dans une liste blanche.

Si l'émetteur est de confiance, le serveur positionne dans la réponse l'en-tête `Access-Control-Allow-Origin` en indiquant comme valeur le domaine de l'émetteur récupéré de la requête. Le navigateur, s'il est "CORS compliant" (c'est le cas de tous les navigateurs récents, voir <http://caniuse.com/#feat=cors> ), acceptera la requête cross domaines et restituera la valeur de la réponse au script.

Si l'émetteur n'est pas de confiance, le serveur a le choix de ne rien renvoyer ou de renvoyer une réponse en erreur avec un code http différent de 2xx.

### – Exemple de Requête :

```
GET /employees/9565841245 HTTP/1.1
Host: credit-agricole.fr
...
Origin: https://credit-agricole.fr
```

### – Exemple de Réponse :

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
...
Access-Control-Allow-Origin: https://credit-agricole.fr
```

#### 5.2.1.3 Désérialisation des données JSON sous contrôle de l'utilisateur

Certaines valeurs transportées dans les échanges peuvent masquer du code malicieux. Il est donc indispensable de ne pas utiliser les valeurs directement. Par exemple, ne jamais insérer de données JSON dans le Document Object Model (DOM) via la fonction JavaScript `.innerHTML`

### – Exemple (à ne pas suivre)

```
document.getElementById("address").innerHTML = JSON.parse(data) ["address"];
```

On utilisera par exemple une des fonctions suivantes : `.value`, `.innerText` ou `.textContent`.

### – Exemple

```
document.getElementById("address").value= JSON.parse(data) ["address"];
```

#### 5.2.1.4 Gestion des types de contenus échangés

Il est indispensable de maîtriser le type de contenu transmis dans les échanges :

- Côté client : Toujours renseigner un en-tête `Content-Type` dans les requêtes.
- Côté serveur :
  - Contrôler que le type de contenu envoyé par le client correspond à l'entête `Content-Type` de la requête.  
Répondre par un code d'erreur `400 Bad Request` si ce n'est pas le cas.
  - Toujours positionner l'entête `Content-Type` dans les réponses, avec la valeur correspondant au contenu restitué.

- Positionner l'entête `X-Content-Type-Options: nosniff` pour protéger les anciens navigateurs des attaques s'appuyant sur le type mime. Cet en-tête empêche notamment Internet Explorer d'interpréter le type de contenu.
- Positionner l'entête `X-Frame-Options: deny` pour protéger les anciens navigateurs des attaques de type "clickjacking" (voir <https://en.wikipedia.org/wiki/Clickjacking> pour plus le détail sur ce type d'attaque).

### 5.2.2 Protection des données échangées

Afin de garantir l'intégrité et la confidentialité des données échangées lors de l'émission d'une requête vers une ressource, qu'il s'agisse de données métier, techniques ou de sécurité, les mesures suivantes doivent être mises en œuvre.

A noter que concernant la protection des données liées à l'authentification et l'autorisation de l'application consommatrice et de l'utilisateur, il faut se référer au §5.3.

#### 5.2.2.1 Chiffrement TLS entre le client d'appel et le serveur qui héberge l'API ou la ressource

Cette mesure induit le déploiement d'un certificat serveur sur le serveur qui expose des API et le contrôle des certificats par le navigateur, l'application ou le serveur qui réalise l'appel.

#### 5.2.2.2 Obligation du navigateur à utiliser TLS

**Cette mesure ne concerne que les cas d'usage où les appels sont issus d'un navigateur.**

Il s'agit de mettre en œuvre **sur le serveur qui sert les pages de l'application consommatrice**, l'en-tête `Strict-Transport-Security` (HSTS – voir [HSTS] pour plus de détails).

```
Strict-Transport-Security: max-age=expireTime  
    [; includeSubDomains] [; preload]
```

- `expireTime`: spécifie, en secondes, la période pendant laquelle le navigateur utilisera exclusivement https pour accéder au site.

#### 5.2.2.3 Chiffrement des données sensibles produites au niveau du client d'appel ou du serveur qui héberge le service.

**Cette mesure concerne les échanges portant sur des données dont la confidentialité est élevée (C=4 ou éventuellement 3, à apprécier dans le contexte du projet).**

Dans ce cas, l'utilisation d'un canal sécurisé TLS n'est pas suffisante pour garantir la confidentialité des données car des équipements intermédiaires pratiquant le déchiffrement TLS peuvent y accéder.

Elle pourra être mise en œuvre côté consommateur pour chiffrer **de manière applicative** les données sensibles à transmettre, ou bien côté fournisseur pour chiffrer les données sensibles à restituer.

**Seules les données sensibles sont chiffrées de manière obligatoire. Si ces données sont comprises dans une structure plus globale qui comprend des données non sensibles, les données non sensibles ne seront pas nécessairement chiffrées.**

Dans les deux cas l'entité qui reçoit les données chiffrées doit contrôler la validité du certificat contenant la clé publique utilisée pour chiffrer (voir 5.2.4.2 pour le détail des contrôles à réaliser sur un certificat).

**Le chiffrement doit toujours être réalisé sur un serveur.**

Si le chiffrement concerne des données sensibles situées sur un poste de travail, ce dernier devra appeler dans son entité d'appartenance un service qui réalisera le chiffrement.

**Remarque :** Les modalités d'appel de ce service sont à la main de l'entité car il ne s'agit pas d'un appel de service inter-entités.

Voir 5.2.3 pour la mise en œuvre technique du chiffrement.

### **5.2.2.4 Chiffrement des données sensibles échangées de serveur à serveur qui transitent par un terminal utilisateur (poste de travail, mobile, ...)**

**Cette mesure concerne les échanges portant sur des données dont la confidentialité est élevée (C=4 ou éventuellement 3, à apprécier dans le contexte du projet).**

Dans ce cas, l'utilisation d'un canal sécurisé TLS n'est pas suffisante pour garantir la confidentialité des données car des équipements intermédiaires pratiquant le déchiffrement TLS peuvent y accéder.

**Seules les données sensibles sont chiffrées de manière obligatoire. Si ces données sont comprises dans une structure plus globale qui comprend des données non sensibles, les données non sensibles ne seront pas nécessairement chiffrées.**

L'entité qui reçoit les données chiffrées doit contrôler la validité des éléments transmis ayant participé au chiffrement.

### **5.2.2.5 Signature des données sensibles produites au niveau du client d'appel ou du serveur qui héberge l'API**

Cette mesure concerne les données collectées sur le poste de travail ou restituées par le serveur lors de la réponse.

**Elle n'est pas obligatoire. Elle concerne les échanges portant sur des données dont la confidentialité est élevée (C=4 ou éventuellement 3, à apprécier dans le contexte du projet).**

Elle pourra être mise en œuvre côté consommateur pour signer les données sensibles à transmettre, ou bien côté fournisseur pour signer les données sensibles à restituer. Dans les deux cas l'entité qui reçoit les données signées doit contrôler la validité de la signature (voir 5.2.4.3 pour le détail des contrôles à réaliser sur une signature).



### 5.2.2.6 *Signature des données sensibles échangées de serveur à serveur qui transitent par un poste de travail*

**Cette mesure est obligatoire** dès lors que l'exigence d'intégrité des données transmises est supérieure à 1. Elle concerne tous les types de données échangées lors de l'accès à une ressource (notamment les données de sécurité).

Le format JWS (JSON Web Signature) est préconisé si les données sont transmises au format JSON, (voir [\[JWS\]](#) pour plus de détail). JWS (RFC 7515) va permettre de créer une signature numérique du message.

L'entité qui reçoit les données signées doit contrôler la validité de la signature (voir 5.2.4.3 pour le détail des contrôles à réaliser sur une signature).

Cette mesure s'applique aux données contenues dans les jetons de sécurité (Access Token, etc.), mais aussi aux données fonctionnelles (ou applicatives) transmises dans le BODY de la requête (ou l'URI ou la QueryString). Ainsi, lorsque le cas se présente, une requête de lancement d'une API peut contenir un jeton Access Token (pour autoriser l'accès à l'API) et un 2<sup>ème</sup> jeton JWT applicatif (signé et, éventuellement chiffré) dans le BODY de la requête.

### 5.2.3 *Chiffrement des données sensibles*

Lorsque les échanges portent sur des données dont la confidentialité est élevée (C=4 ou éventuellement 3, à apprécier dans le contexte du projet), ces données sensibles doivent être chiffrées. Le format JSON Web Encryption (JWE) est préconisé si les données sont transmises au format JSON, (voir [\[JWE\]](#) pour plus de détail).

Cela ne constitue toutefois pas une obligation. Si une entité choisit d'utiliser une solution de chiffrement différente, elle devra assumer le risque résiduel porté par la solution utilisée. A titre d'exemple, dans certain cas d'usage, il peut être jugé acceptable de signer l'ensemble des données via JWS et de ne chiffrer que les données les plus sensibles via un algorithme de chiffrement symétrique (type AES256). Si le risque résiduel associé à l'utilisation d'une clé symétrique est accepté, cela a l'avantage de simplifier la réalisation et la maintenance de l'échange (pas de certificat de chiffrement à renouveler).

Les projets ont toute latitude pour ce qui concerne les optimisations pour le chiffrement comme le regroupement dans une structure et le chiffrement en une fois de différentes données sensibles initialement contenues dans des structures différentes.

A noter que cette mesure s'applique aux données sensibles contenues dans les jetons de sécurité (Access Token, etc.), mais aussi aux données fonctionnelles (ou applicatives) sensibles transmises dans le BODY de la requête (ou l'URI ou la QueryString). Ainsi, lorsque le cas se présente, une requête de lancement d'une API peut contenir un jeton Access Token (pour autoriser l'accès à l'API) et un 2<sup>ème</sup> jeton JWT applicatif (signé et, éventuellement chiffré) dans le BODY de la requête.

### 5.2.3.1 Format technique du jeton JWE

Le format technique retenu utilise la "JWE Compact Serialization".

Un jeton JWE est constitué de la concaténation de 5 objets JSON (un header, une EncryptedKey, un vecteur d'initialisation, les données chiffrées et une balise d'authentification) encodés en base64url et séparés par des points :

Jeton JWE = BASE64URL(UTF8(JWE Protected Header)) || '.' || BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE Initialization Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE Authentication Tag)

- L'en-tête (JWE Protected Header) est constitué comme suit :

```
{  
  "alg": "RSA-OAEP-256",  
  "enc": "A256GCM",  
  "zip": "DEF",  
  "x5c": "[un certificat X509]",  
  "kid": "[DN du certificat X509 utilisé pour chiffrer le jeton]",  
  "cty": "JWT"  
}
```

- **alg** : spécifie l'algorithme de chiffrement asymétrique utilisé (RSA-OAEP-256) pour crypter la clé de chiffrement des données applicatives (Content Encryption Key - CEK) ;
- **enc** : spécifie l'algorithme symétrique de chiffrement utilisé pour crypter les données utiles et produire le "ciphertext" et l'"Authentication Tag" (A256GCM = algorithme AES GCM avec une clé de chiffrement symétrique de 256 bits) ;
- **zip** : spécifie l'algorithme de compression appliqué sur les données applicatives à chiffrer avant chiffrement. "DEF" correspond à compression avec l'algorithme DEFLATE (RFC1951 voir <http://tools.ietf.org/html/rfc1951>) ;
- **x5c** : contient le certificat (ou la chaîne de certificats) qui comprend la clé publique du destinataire utilisée pour chiffrer la clé de chiffrement (variante autorisée : **x5u** qui comprend une uri vers cette clé publique) ;
- **kid** (Key identifier) : correspond au DN du certificat ayant servi à chiffrer. Cette donnée est facultative.
- **cty** : correspond au content type du jeton. Il doit être renseigné avec la valeur "JWT" car il englobe un autre jeton JWT.
- **EncryptedKey** : la clé symétrique de chiffrement utilisée pour le chiffrement des données applicatives, chiffrée avec la clé publique du destinataire.
- **Initialization\_Vector** : une clé aléatoire qui sera le vecteur d'initialisation nécessaire dans l'algorithme de chiffrement symétrique.
- **Ciphertext** : les données applicatives chiffrées.

Le format des données utiles à chiffrer peut varier :

- S'il s'agit de données sensibles collectées sur le poste ou restituées lors de l'accès à une ressource, la structure (objet JSON, message XML, autre) sera à définir dans le projet concerné ;

- S'il s'agit d'un jeton que l'on souhaite simultanément chiffrer et signer, il est préconisé dans la spécification JWT (voir <https://tools.ietf.org/html/rfc7519#section-11.2>) de produire dans un premier temps un jeton signé (JWS) et de l'inclure comme données utiles à chiffrer dans le jeton chiffré (JWE).
- **Authentication Tag** : la signature RSA du header, concaténée avec les données applicatives (ciphertext).

### 5.2.4 Contrôles à réaliser sur les éléments échangés

#### 5.2.4.1 Principes généraux

**Lorsqu'elle reçoit des données chiffrées ou signées, une entité impliquée dans les échanges doit de manière obligatoire vérifier systématiquement la validité des composants ayant permis la signature ou le chiffrement : certificat et signature électronique.**

Cela concerne le serveur qui héberge une ressource, une API, et reçoit un jeton de sécurité.

Cela concerne également l'application qui accède à la ressource, à l'API, lorsqu'elle reçoit dans la réponse à la requête des données sensibles.

Le serveur qui héberge une ressource, une API, doit à minima vérifier :

- La validité du jeton de sécurité en s'assurant que, sur le serveur sur lequel on réalise le contrôle, la date à laquelle on réalise le contrôle est située dans l'intervalle compris entre "iat" et "exp".

**Le délai entre la date de création et la date d'expiration est à fixer au niveau des projets. Il ne fait pas l'objet d'une norme.**

Néanmoins on gardera à l'esprit que plus ce délai est long et plus la probabilité de vol ou de corruption des données est grande.

- La validité du « jti » (si jeton à usage unique)
- Les données d'identification et d'authentification de l'application appelante
- Les droits de l'utilisateur
- L'adéquation des données d'authentification de l'utilisateur à la criticité du service appelé

Il s'agit de vérifier la cohérence entre la sensibilité de la ressource par rapport au risque d'une utilisation malveillante et le "niveau" de jeton présenté. Le niveau fait correspondre la durée de validité du jeton aux caractéristiques de l'authentification de l'utilisateur (heure de l'authentification, force de l'authentification et méthode d'authentification).

Si le contrôle n'est pas satisfait, le serveur renvoie une erreur optionnellement accompagnée d'un message indiquant le niveau minimum requis pour le jeton. Cette disposition est de la responsabilité de l'entité fournisseur de l'API.

Il est également nécessaire de restituer à l'application appelante les informations qui lui permettront d'initier le processus de réauthentification de l'utilisateur.

Si tous les contrôles de sécurité sont satisfaits, l'API appelée réalise le traitement et l'accès aux ressources. Les contrôles de validité des paramètres fonctionnels ou des données métier transmis

pourront également donner lieu à des erreurs, la réponse devra être conforme aux spécifications décrites dans les chapitres correspondant aux différentes requêtes.

### 5.2.4.2 Validation d'un certificat

Les contrôles à réaliser sont les suivants :

- Vérifier que le certificat n'est pas expiré.
- Vérifier que la chaîne d'Autorité de Certification (AC) émettrice du certificat n'est pas expirée.
- Vérifier que la chaîne d'AC émettrice du certificat est connue.
- Vérifier que le certificat n'est pas révoqué par la chaîne d'AC émettrice.
- Vérifier la validité de la signature de la clé publique transmise dans le certificat par la clé publique de l'AC émettrice.

### 5.2.4.3 Vérification d'une signature

**Cette mesure est obligatoire.** Les contrôles à réaliser sont les suivants :

- Contrôler conformément à la mesure précédente le certificat contenant la clé publique associée à la clé privée utilisée pour signer.
- Dans le cas où le jeton est signé et, éventuellement chiffré, par l'application consommatrice, il faut alors contrôler que le serveur qui a signé le jeton est bien habilité à le faire (cette mesure de sécurité ne s'applique pas aux jetons de sécurité produit dans le cadre OAuth/OpenID Connect décrit au §5.3). Pour ce faire, il est nécessaire de gérer une liste blanche qui contient les identifiants des serveurs autorisés à signer.
  - La liste blanche associée à une ressource contient l'ensemble des serveurs autorisés à générer un jeton signé pour cette ressource.
  - Chaque serveur est identifié par un Distinguished Name (DN) conforme à la RFC 5280 (description d'un certificat X509) Chap. 4.1.2.6 (<https://tools.ietf.org/html/rfc5280#section-4.1.2.6>).
  - Le DN doit être caractéristique d'un serveur dans un environnement et dans une entité. Un certificat ne peut être partagé ni entre entités, ni entre environnements dans une entité.
  - Lors de l'accès à une ressource, le fournisseur contrôle la présence dans la liste blanche du subject Distinguished Name indiqué dans le certificat qui a servi pour signer le jeton.

Le consommateur de la ressource doit au préalable transmettre au fournisseur le DN pour qu'il puisse alimenter la liste blanche. Le DN est au format texte.

  - Le contrôle est fait attribut par attribut sur tous les attributs du DN.
  - Le fournisseur s'assurera que le DN est bien relatif à une entité, un environnement et un serveur.

- La liste blanche doit être externalisée pour pouvoir être modifiée facilement. L'implémentation est à la main du fournisseur.
- Il est nécessaire de permettre le débrayage du contrôle.
- Contrôler que la signature est valide :
  - Produire un condensat (hash) des données utiles en utilisant la fonction de hachage indiquée ;
  - Comparer avec la signature fournie après déchiffrement.

La vérification de la signature pourra être réalisée en utilisant un composant de marché.

### 5.2.5 Gestion des certificats

Dans le cadre de la présente norme, 2 typologies de certificat sont nécessaires : les certificats TLS pour sécuriser les accès HTTP et les certificats logiciels pour sécuriser les jetons (JWT, etc.).

**Pour la gestion des certificats TLS destinés aux sites web visibles du public (client, etc.), il est préconisé d'utiliser l'offre Groupe qui s'appuie sur une PKI externe dont les tarifs ont été négociés pour le Groupe. Cette offre Groupe est gérée par CAGIP/ADSI.**

**Pour la gestion des certificats TLS destinés aux sites web visibles uniquement des collaborateurs du Groupe, il est préconisé d'utiliser les services de la PKI Groupe (Carioca).**

**Pour la gestion des certificats logiciels, il est préconisé d'utiliser les services de la PKI Groupe (Carioca) qui propose des certificats logiciels (pour la signature et le chiffrement applicatif JWS/JWE).**

Lien vers la page CARIOCA de l'Intranet CASA/SRI :

[https://ca-sa.ca-mocca.com/site/ssi/fr-fr/int%C3%A9grer\\_s%C3%A9curit%C3%A9/Pages/CARIOCA.aspx](https://ca-sa.ca-mocca.com/site/ssi/fr-fr/int%C3%A9grer_s%C3%A9curit%C3%A9/Pages/CARIOCA.aspx)

La documentation CARIOCA est disponible à l'adresse suivante :

[https://service.ca-mocca.com/maia-collab/ssi/Services\\_CARIOCA/01\\_Version%20Fran%C3%A7aise](https://service.ca-mocca.com/maia-collab/ssi/Services_CARIOCA/01_Version%20Fran%C3%A7aise)

Une gouvernance, s'assurant de la coordination des actions au niveau groupe a été mise en place. Elle est animée par Crédit Agricole S.A./SIG/SRI. Dans chaque entité du groupe, un correspondant pour la gestion des certificats a été nommé.

#### 5.2.5.1 Mise à disposition des certificats logiciels via URL

Les certificats logiciels sont utilisés dans la présente norme pour la signature ou le chiffrement de jeton JWT.

**Dans le cadre d'une signature de jeton**, le certificat de signature utilisé par l'application source devra être mis à disposition de l'applicatif cible (API) pour qu'il puisse vérifier la signature (voir la liste des contrôles à effectuer au « §5.2.4.3 Vérification de la signature »).

Pour le **certificat de signature d'un jeton JWT**, il est **préconisé de le mettre à disposition** de l'applicatif cible, soit **en tant qu'attribut du jeton** (dans ce cas le jeton est autoporteur du certificat), soit **en le rendant accessible via URL**.

Dans le cadre d'un chiffrement de jeton par une application source, à destination d'un applicatif cible, le certificat de chiffrement devra être mis à disposition par l'applicatif cible.

Pour le **certificat de chiffrement d'un jeton JWT**, il est **préconisé que l'applicatif cible le mette à disposition** de l'application source, **via URL**.

Ainsi, ces mises à disposition dynamique des certificats logiciels (via URL) permettront de simplifier grandement le renouvellement des certificats, en limitant l'impact à l'application propriétaire du certificat.

Lorsque le certificat est mis à disposition via URL, il devra être :

- Accessible sans contrôle d'accès à toutes les applications souhaitant y accéder (ouvertures de flux effectuées)
- Uniquement accessible en HTTPS
- Au format JWKS (JSON Web Key Set) – RFC7517 (<https://tools.ietf.org/html/rfc7517>)
- Positionner sur un serveur web dont le niveau de disponibilité sera en correspondance avec le besoin des applications utilisatrices de ce certificat (donc, en générale, en haute disponibilité).
- Exemple d'URL :

**`https://myapp.entite1.group.gca/myapp-certs.jwks`**

Côté applicatif utilisateur de ce certificat, il est préconisé de conserver ce certificat en cache, pour ne pas avoir à le récupérer systématiquement et pour ne pas provoquer d'incident si l'URL du certificat est temporairement indisponible. La durée du cache sera à définir en fonction de l'analyse de risque de l'application.

## 5.3 Authentification et autorisation de l'application consommatrice et de l'utilisateur pour l'accès à une API

Ce chapitre décrit les spécifications pour la sécurisation des accès aux API.

### 5.3.1 OAuth2 et OpenID Connect (OIDC)

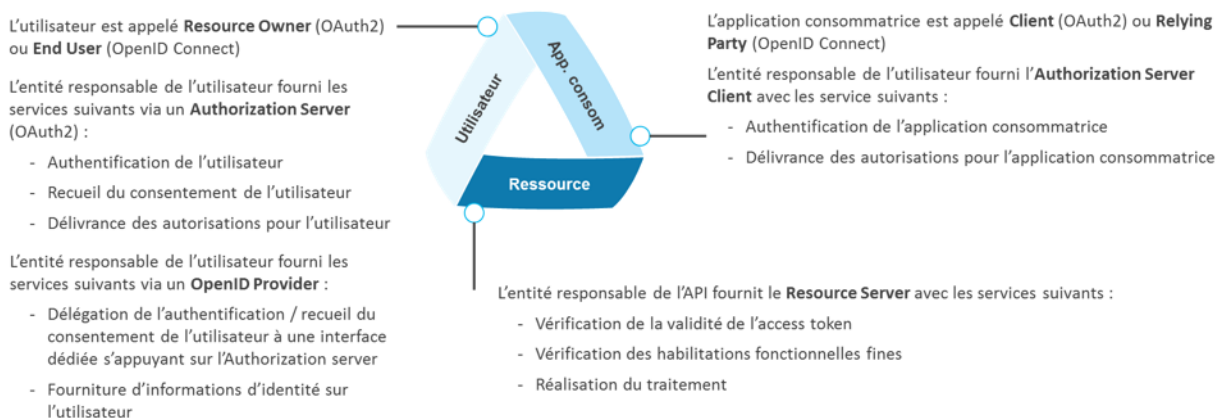
Pour favoriser l'interopérabilité au sein du Groupe et accélérer la mise à disposition des ressources aux consommateurs externes (Open API, etc.), la sécurisation des API s'appuie sur les standards de marché OAuth2 et OpenID Connect. Cela implique la bonne application des standards suivants :

Libellé	Code	Référence	Application dans le contexte Groupe CA
---------	------	-----------	--

JSON Web Token (JWT)	RFC7519	<a href="https://tools.ietf.org/html/rfc7519">https://tools.ietf.org/html/rfc7519</a>	Oui
JSON Web Signature (JWS)	RFC7515	<a href="https://tools.ietf.org/html/rfc7515">https://tools.ietf.org/html/rfc7515</a>	Oui
JSON Web Encryption (JWE)	RFC7516	<a href="https://tools.ietf.org/html/rfc7516">https://tools.ietf.org/html/rfc7516</a>	Oui
OAuth 2.0 Core	RFC6749	<a href="https://tools.ietf.org/html/rfc6749">https://tools.ietf.org/html/rfc6749</a>	Oui (3 Grant) Authorization Code Grant Resource Owner Password Credentials Grant Client Credentials Grant
JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens	RFC9068	<a href="https://datatracker.ietf.org/doc/html/rfc9068">https://datatracker.ietf.org/doc/html/rfc9068</a>	Oui
OAuth 2.0 Bearer Token Usage	RFC6750	<a href="https://tools.ietf.org/html/rfc6750">https://tools.ietf.org/html/rfc6750</a>	Oui
OAuth 2.0 JWT Profile for Client Authentication and Authorization Grants	RFC7523	<a href="https://tools.ietf.org/html/rfc7523">https://tools.ietf.org/html/rfc7523</a>	Oui
OAuth 2.0 for Native Apps	RFC8252	<a href="https://tools.ietf.org/html/rfc8252">https://tools.ietf.org/html/rfc8252</a>	Oui
OAuth 2.0: Audience Information		<a href="https://tools.ietf.org/id/draft-tschofenig-oauth-audience-00.html">https://tools.ietf.org/id/draft-tschofenig-oauth-audience-00.html</a>	Oui
OAuth 2.0: Token Revocation	RFC7009	<a href="https://tools.ietf.org/html/rfc7009">https://tools.ietf.org/html/rfc7009</a>	Oui
OAuth 2.0 Token Introspection	RFC7662	<a href="https://tools.ietf.org/html/rfc7662">https://tools.ietf.org/html/rfc7662</a>	
OAuth 2.0 Token Exchange	RFC8693	<a href="https://datatracker.ietf.org/doc/html/rfc8693">https://datatracker.ietf.org/doc/html/rfc8693</a>	Oui
OAuth 2.0 Resource Indicators	RFC8707	<a href="https://datatracker.ietf.org/doc/html/rfc8707">https://datatracker.ietf.org/doc/html/rfc8707</a>	Oui
Authentication Method Reference Values (AMR)	RFC8176	<a href="https://tools.ietf.org/html/rfc8176">https://tools.ietf.org/html/rfc8176</a>	Oui
Proof Key for Code Exchange by OAuth Public Clients (PKCE)	RFC7636	<a href="https://tools.ietf.org/html/rfc7636">https://tools.ietf.org/html/rfc7636</a>	Oui
OpenID Connect Core 1.0	openid-connect-core-1_0	<a href="http://openid.net/specs/openid-connect-core-1_0.html">http://openid.net/specs/openid-connect-core-1_0.html</a>	Oui (1 Flow) Authorization Code Flow

## 5.3.2 Rôles

La correspondance des rôles décrits dans le paragraphe 2.1.3 avec ceux portés dans les normes OAuth2 et OpenID Connect est illustrée dans le schéma ci-dessous :



**Remarque :** dans le cas des accès collaborateurs, si l'utilisateur authentifié via l'application consommatrice ne correspond pas au propriétaire de la ressource appelée (ex : compte bancaire d'un client). L'autorisation accordée par le client de la Banque au collaborateur de gérer ses données est considérée comme implicite à la signature de la convention de compte et se matérialise dans les règles d'habilitations du collaborateur. Ainsi la demande de consentement n'est pas nécessairement



demandée dans les cinématiques OAuth2 / OIDC. Il en est de même lorsqu'un client de la Banque accède à ses données bancaires via un canal couvert par sa convention de compte.

## 5.3.3 Types de jetons intervenants dans OAuth et OpenID Connect

Le tableau ci-dessous décrit les 4 types de jetons de sécurité véhiculés dans les échanges, selon la cinématique OAuth2 / OpenID Connect utilisée. D'autres jetons applicatifs (i.e. données applicatives signées, et éventuellement, chiffrées) peuvent également être échangés en plus.

	Description
Access Token (AT)	<p><b>L'Access Token est le jeton permettant d'autoriser l'accès d'une application consommatrice à une API</b></p> <ul style="list-style-type: none"> <li>- Son format n'est pas complètement spécifié dans OAuth2, mais deux cas sont principalement utilisés : <ul style="list-style-type: none"> <li>- Format JWT (JSON Web Token) : pour un jeton autoporteur. L'ensemble des données à destination de l'API et de la ressource sont présentes dans le jeton.</li> <li>- Format UUID : pour un jeton « opaque ». Ici, le jeton n'est qu'un simple identifiant UUID et ne porte donc aucune donnée. Les données associées à ce jeton restent stockées au niveau de l'Authorization Server et ainsi ne circulent pas sur le réseau et ne sont pas accessibles de l'application consommatrice.</li> </ul> </li> <li>- Il s'agit d'un jeton porteur de l'autorisation, d'une durée de vie définie (en fonction du besoin métier et de l'analyse de risque), utilisé pour contrôler l'accès aux ressources. <ul style="list-style-type: none"> <li>- Il doit avoir une durée de vie au moins égale à une action utilisateur sur l'interface (lors d'un clic sur l'interface, toutes les actions effectuées doivent pouvoir s'effectuer à l'aide d'un même access token)</li> </ul> </li> <li>- Par défaut il est réutilisable pendant sa durée de validité (pas de notion transactionnelle définie)</li> <li>- Il est également réutilisable pour l'accès aux différentes API contenues dans l'attribut « audience » du jeton</li> <li>- Une entité peut choisir d'imposer un usage unique du jeton pour une API donnée, en se basant sur son identifiant technique. C'est l'attribut « jti » dans l'Access Token).</li> <li>- Quelle que soit la cinématique retenue, l'Access Token ne peut véhiculer que des données validées par l'Authorization Server en charge de sa génération.</li> <li>- Dans un format JWT, lorsqu'il est délivré à des applications consommatrices externes au Groupe, il est recommandé d'opter pour un jeton chiffré de manière à ne pas rendre son contenu accessible à l'application consommatrice. Ce choix est à discrétion de l'entité en charge de son émission, en fonction de l'analyse de risque associée à l'usage.</li> </ul>
Refresh token (RT)	<p><b>Le Refresh Token représente un contexte utilisateur (issu d'une délégation d'autorisation par un utilisateur), qui peut être utilisé même en son absence. Son rôle est uniquement de permettre le renouvellement de l'Access Token initiale, lui correspondant.</b></p> <ul style="list-style-type: none"> <li>- Son format n'est pas spécifié dans OAuth2 mais l'état de l'art le définit comme un UUID ne contenant aucune donnée fonctionnelle. Il peut également s'agir d'un jeton JWT autoporteur.</li> <li>- Il a une durée de vie définie en fonction du besoin métier et de l'analyse de risque sécurité</li> <li>- Il est comparable à un cookie de session sur une application Web et peut être utilisé pour obtenir un Access Token correspondant à l'authentification initiale</li> </ul>



	<ul style="list-style-type: none"> <li>- Il pourra être révoqué par l'application si aucun accès de l'application n'est nécessaire après déconnexion de l'utilisateur</li> <li>- En application du pattern « Privacy By Design », ce jeton n'est pas systématiquement délivré à l'application consommatrice (dépend de la cinématique et du caractère sûr ou non sûr de l'application)</li> </ul>
<b>Authorization code (AZ)</b>	<p><b>L'Authorization Code n'est utilisé que dans le cas de la cinématique OAuth2 du même nom</b></p> <ul style="list-style-type: none"> <li>- Il a pour fonction de partager sur un support non sûr une preuve temporaire et à usage unique de l'authentification. Sa durée de validité est de l'ordre de quelques minutes et il est recommandé de ne pas dépasser les 10 minutes de validité (RFC6749).</li> <li>- Il est en particulier utilisé dans le cas d'application mobile pour permettre l'échange de la preuve d'authentification entre une interface de confiance et une application non sûre. OAuth2 ne spécifie pas son format. Dans le cadre de la présente norme API, nous préconisons l'utilisation du format UUID (Universal Unique Identifier)</li> </ul>
<b>ID Token (ID)</b>	<p><b>L'ID Token est un jeton signé permettant de retourner à l'application consommatrice la preuve d'authentification de l'utilisateur et ses informations d'identité.</b></p> <ul style="list-style-type: none"> <li>- Il est spécifié dans la norme OpenID Connect. Il est au format JWT avec un certain nombre de champs obligatoires et optionnels déjà définis par la norme.</li> <li>- L'ID Token est destiné en premier lieu à l'application consommatrice (i.e. application cliente). Il n'a pas vocation à être transmis aux API (c'est l'Access Token qui le sera). Ainsi, l'attribut « audience » de l'ID Token devra contenir, au minimum, le client_id de l'application consommatrice.</li> <li>- Dans certain cas d'usage, l'ID Token pourra être transmis par l'application consommatrice à une application tierce comme preuve d'authentification de l'utilisateur (SSO). Dans ce cas, l'attribut « audience » de l'ID Token devra contenir les identifiants de ces applications tierces. L'alimentation de l'audience de l'ID Token est faite par l'Authorization Server (ou OpenID Provider) sur la base des identifiants d'applications tierces déclarées par l'application cliente lors de son enrôlement.</li> <li>- En application du pattern « Privacy By Design » : <ul style="list-style-type: none"> <li>- Ce jeton est délivré, de manière facultative et à la demande de l'application consommatrice (via le scope « openid »), sous réserve que celle-ci soit autorisée à le faire.</li> <li>- Ne sont retournées dans ce jeton que les informations demandées dans le scope (cf. valeurs prédéfinies du scope : <a href="#">cf. 5.3.5.6 Glossaire des données utilisées dans les jetons</a>) par l'application consommatrice et autorisées par le serveur d'autorisation.</li> </ul> </li> </ul>

## 5.3.4 Méthodes d'authentification des applications consommatrices clientes

### 5.3.4.1 Choix de la méthode d'authentification des applications

Conformément au standard OAuth 2, quel que soit la cinématique OAuth choisie (§5.3.5), **les applications consommatrices d'API (« Client ») disposent de plusieurs méthodes pour s'authentifier sur l'Authorization Server. Il conviendra de choisir la méthode la mieux adaptée au contexte et, notamment, aux besoins de sécurité définis dans l'analyse de risque de l'application.**

La méthode choisie pourra être indiquée lors de l'enrôlement de l'application sur le portail consommateur. Par défaut, si aucune méthode n'est renseignée, la méthode « client\_secret\_basic »

sera appliquée. C'est-à-dire que l'application s'authentifiera avec son `client_id` et `client_secret`, via une « HTTP Basic authentication ».

Les différentes méthodes d'authentification des applications consommatrices sur l'Authorization Server sont décrites dans le standard OpenID Connect ([lien](#)) et OAuth ([rfc7523](#) pour l'authentification via JWT), et synthétisées ci-dessous (de la moins sécurisée à la plus sécurisée) :

Méthode d'authentification	Description
<b>none</b>	Aucune authentification de l'application consommatrice d'API (Client)
<b>client_secret_basic</b> (méthode par défaut)	Authentification avec le <code>client_id</code> et <code>client_secret</code> de l'application, reçus lors de la phase d'enrôlement. Cette authentification s'effectuera via une « HTTP Basic Authentication ».
<b>client_secret_post</b>	Authentification avec le <code>client_id</code> et <code>client_secret</code> de l'application, reçus lors de la phase d'enrôlement. Le <code>client_id</code> et <code>client_secret</code> seront positionnés dans le <i>body</i> de la requête d'authentification et envoyés en <i>POST</i> .
<b>client_secret_jwt</b>	Authentification via un JSON Web Token créé avec l'algorithme « HMAC SHA algorithm » avec un HMAC produit à partir du <code>client_secret</code> reçu par l'application lors de son enrôlement
<b>private_key_jwt</b>	Authentification via un JSON Web Token signé par cryptographie asymétrique grâce à un couple clé public (certificat)/clé privée. Le certificat sera communiqué à l'Authorization Server lors de la phase d'enrôlement de l'application.

**Chacune de méthodes est applicable à toutes les cinématiques OAuth.**

Remarque : si une solution progiciel est utilisée pour l'Authorization Server (WSO2, Keycloak, etc.), il faudra vérifier que la méthode d'authentification choisie est bien prise en charge. Pour information, WSO2 et Keycloak sont, au minimum, compatibles avec les méthodes « none », « client\_secret\_basic », « private\_key\_jwt », qui sont les 3 méthodes que la norme préconisera dans les cinématiques OAuth.

### 5.3.4.2 Mise en œuvre des méthodes d'authentification des applications

Dans la cinématique OAuth, l'authentification de l'application Client est effectuée lors de l'**Access Token Request**, la **Refresh Token Request** ou la **Revoke Token Request** (voir les cinématiques décrites au [§5.3.5](#)).

Dans les paragraphes suivants, nous prendrons l'exemple de la cinématique « Client Credentials », la plus simple, pour illustrer les requêtes d'authentification associées aux différentes méthodes. Mais,

bien entendu, les méthodes d'authentification pourront s'appliquer, également aux autres cinématiques (ROPC, AZ Code).

- L'Access Token Request « Client Credentials » avec la méthode « **client\_secret\_basic** »

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64(MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type=client_credentials
```

Conformément au standard OAuth 2 ([RFC6749](#)).

- L'Access Token Request « Client Credentials » avec la méthode « **client\_secret\_post** »

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id= MON_CLIENT_ID
&client_secret= MON_CLIENT_SECRET
```

Conformément au standard OAuth 2 ([RFC6749](#)).

- L'Access Token Request « Client Credentials » avec la méthode « **client\_secret\_jwt** » ou « **private\_key\_jwt** »

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id= MON_CLIENT_ID
&client_assertion_type = "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
&client_assertion = MON_CLIENT_JWT_TOKEN
```

Conformément au standard « JWT Profile for OAuth 2.0 Client Authentication » ([RFC7523](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre « grant\_type » → Obligatoire

- Paramètre « client\_id » → Obligatoire
- Paramètre « client\_assertion\_type » → Obligatoire
  - Type de Jeton d'authentification de l'application consommatrice.
  - Il est utilisé en complément du paramètre « client\_assertion » pour indiquer son type. Pour nos cas d'usage, le « client\_assertion » sera toujours au format JWT, donc, la valeur de client\_assertion\_type sera toujours égale à : "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
- Paramètre « client\_assertion » → Obligatoire
  - Jeton d'authentification de l'application consommatrice.
  - Il est au format JSON Web Token. La signature du JWT pourra s'effectuer soit grâce à une Clé privée / Clé publique (c'est la solution préconisée), soit grâce au « HMAC SHA algorithm » en s'appuyant sur le client\_secret pour générer la HMAC, tel que décrit dans la RFC7523 et OpenID Connect Core 1.0.
  - Les données contenues dans ce jeton JWT d'authentification sont les suivantes (issues de la RFC7523) :
    - Les données standards d'entête JWT : « alg », « cty » et l'attribut identifiant le certificat de signature lorsqu'il est utilisé (« kid » ou « x5c » ou « x5u » ou « x5t#S256 »). A noter que l'Authorization Server cible pourra demander l'URL du certificat ou le certificat lui-même lors de la phase d'enrôlement de l'application.
    - Les données du body du JWT

Intitulé	O/F	Description	Format
iss	O	<b>Émetteur du jeton</b> Doit contenir le <i>client_id</i> de l'application Client	String
sub	O	<b>Identifiant du porteur</b> Doit contenir le <i>client_id</i> de l'application Client	String
aud	O	<b>Audience(s) du jeton</b> Doit contenir l'identifiant de l'Authorization Server cible. L'Authorization Server devra obligatoirement vérifier sa présence dans cette audience.	[String or URI]
exp	O	<b>Heure d'expiration du jeton</b>	Timestamp
iat	O	<b>Heure de délivrance du jeton</b>	Timestamp
jti	O	<b>Identifiant unique du jeton</b>	UUID

### 5.3.5 Choix des cinématiques (ou « Grant Types »)

#### 5.3.5.1 Application du standard au contexte Crédit Agricole

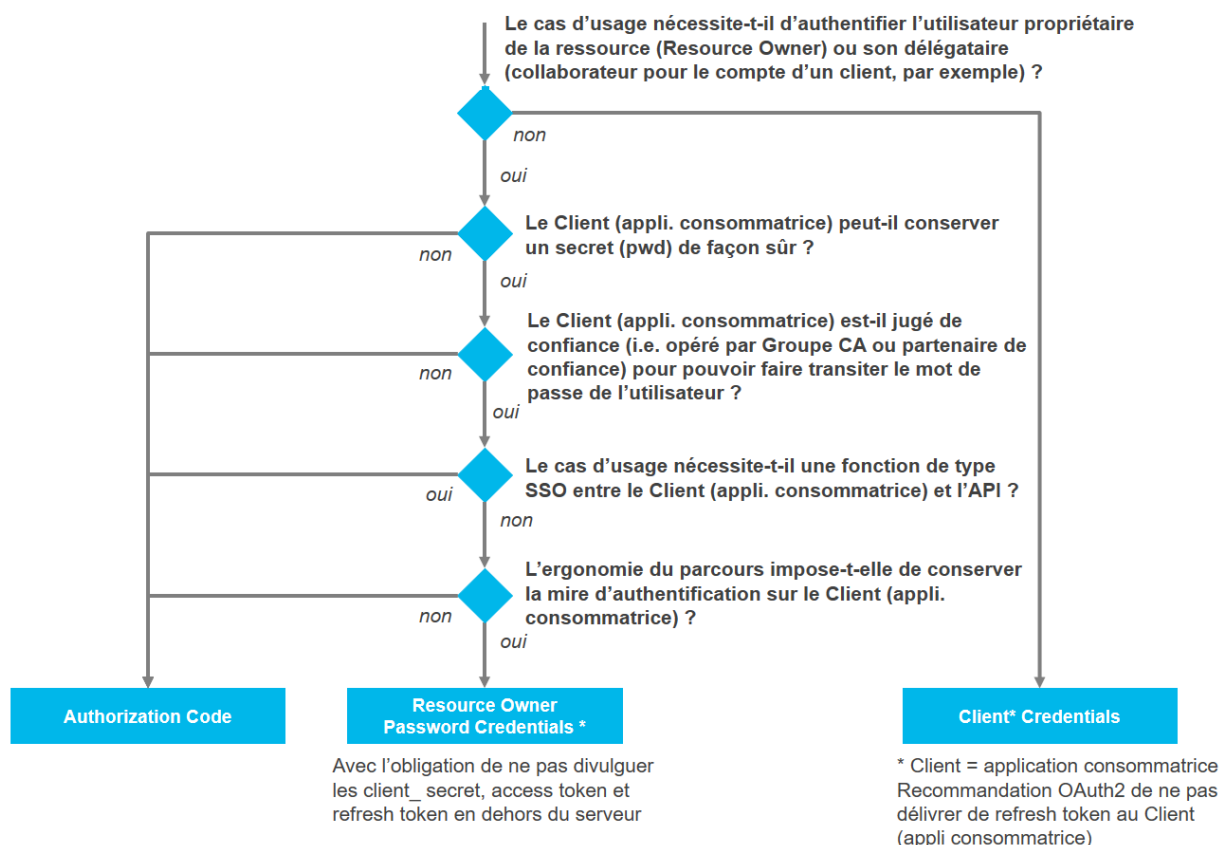
### 5.3.5.1.1 Cas général

**L'application des standards OAuth2 et OpenID Connect au sein du Groupe repose sur l'usage de 3 cinématiques (Grant Types) : Client Credentials, Resource Owner Password Credentials et Authorization Code.**

Le choix de cinématique dépend principalement de deux facteurs :

- **Le besoin d'authentifier un utilisateur** lorsque les ressources consommées le nécessitent pour évaluer des habilitations fonctionnelles, alimenter des pistes d'audit, etc.
- **La nature de l'application consommatrice (i.e. le « Client »)** définie par
  - **Sa capacité à garder un secret**
    - Confidentiel : application pouvant conserver un secret de façon sûre. En général il s'agit d'applications exécutées sur un serveur.
    - Public : application ne pouvant pas conserver un secret de façon sûre. En général il s'agit d'applications natives ou exécutées sur un navigateur.
  - **Son niveau de confiance pour pouvoir faire transiter le mot de passe de l'utilisateur**
    - Sûr : application opérée par le Groupe Crédit Agricole ou un partenaire jugé de confiance.
    - Non sûr : application opérée par une autre entreprise.

**La sélection de la cinématique d'authentification et autorisation (« Grant Types ») à appliquer pour un cas d'usage donné doit être conforme à l'arbre de décision proposé ci-dessous.**



La cinématique utilisée doit être indiquée dans un paramètre « grant\_type » de la requête de demande d'un jeton d'accès (access token request) initiée par l'application Client et à destination de l'Authorization Server

Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

A noter que pour une même API, il est possible d'autoriser plusieurs cinématiques d'authentification et d'autorisation (« Grant Types ») en fonction des applications consommatrices (Client) ou des cas d'usage de l'API.

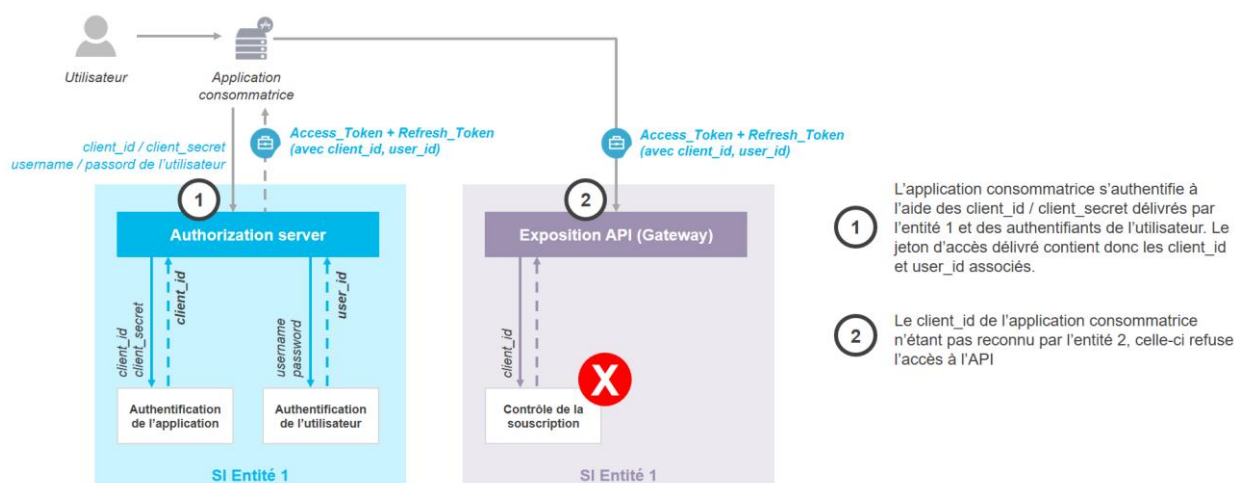
### 5.3.5.1.2 Cas des enrôlements inter-entités

Actuellement, chaque Entité proposant des API implémente un portail consommateur d'enrôlement qui lui est propre. Ainsi, il y a de multiple portails consommateur au sein du Groupe. L'objectif du Groupe est de tendre vers la mise en place d'un portail consommateur et un référentiel d'enrôlement centralisé au niveau Groupe, mais en attendant, la présente norme doit prendre en compte la particularité actuelle.

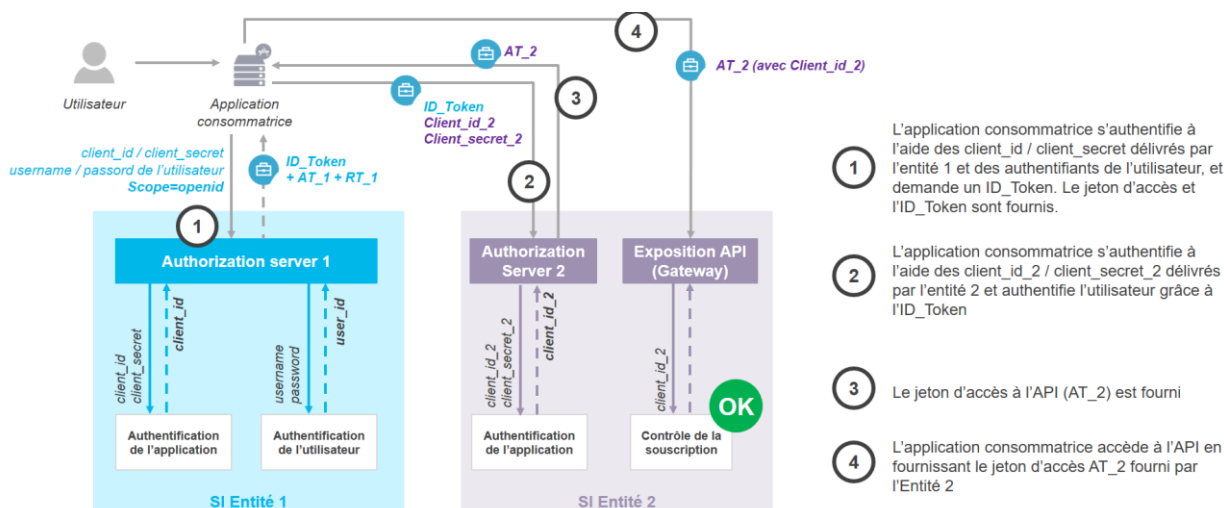
Les standards OAuth2 et OpenID Connect ne couvrent pas nativement les cas d'usage où les client\_id/client\_secret (ou autre authentifiants) issus de l'enrôlement de l'application consommatrice

ne sont pas partagés entre l'Entité en charge de l'authentification de l'utilisateur et l'Entité qui expose la ressource consommée. En effet, en l'état actuel des SI des Entités (un portail d'inscription par Entité), l'application s'inscrivant dans une Entité obtiendra un client\_id/client\_secret qui ne pourra être vérifié que par cette Entité

Le schéma ci-dessous illustre la problématique :



Ainsi, dans les différents cas d'usage inter-entités, nous mettrons en œuvre le jeton ID\_Token du standard OpenID Connect pour transmettre la preuve d'authentification de l'utilisateur (SSO) de l'Authorization Server 1 (Entité 1) vers l'Authorization Server 2 (Entité 2). La cinématique correspondante est illustrée ci-dessous :



A noter que, puisqu'il s'agit d'échanges intra-Groupe, **il existe une confiance réciproque entre les 2 entités concernant l'authentification de l'utilisateur**. Cette confiance est transmise grâce à l'utilisation de certificats issues de la PKI Groupe (Carioca).



De plus, l'**ID\_token** produit par l'Entité 1 **devra contenir l'identifiant** (String ou URI) **de l'Authorization Server 2** (Entité 2) dans son « audience ».

### 5.3.5.1.3 Cas des profils d'habilitation applicatifs inter-entités : scope et user\_profile

Le standard OAuth/OIDC introduit la notion de « **scope** » qui représente le périmètre des droits (habilitations) délégués par l'utilisateur à l'application consommatrice, pour accéder à l'API (et aux données de l'utilisateur). Cette délégation de droits est accordée par l'utilisateur via la validation de son consentement (à noter que, dans le contexte Groupe, la validation du consentement sera implicite pour de nombreux cas d'usage). Ainsi ce « scope » se retrouvera dans les jetons d'accès à une API (Access Token). A noter également, que le paramètre « scope » est alimenté avec des valeurs spécifiques dans le cadre d'une cinématique OpenID Connect

Le modèle Groupe IC08 (cf. [5.1.3 Habilitations fonctionnelles](#)) introduit lui la notion de **profil d'habilitation applicatif** qui est le pivot de correspondance entre les habilitations attribuées à l'utilisateur (côté Entité gérant l'utilisateur) et les permissions d'accès aux ressources (côté fournisseur d'API). Ce profil matérialise l'ensemble des droits accordés à l'utilisateur sur l'API. Ainsi, ce profil d'habilitation applicatif se retrouvera lui dans le jetons JWT applicatif envoyé à une API (donc, en dehors de l'Access Token) avec le nom d'attribut « **user\_profil** ». Voir [§5.1.6](#)

En synthèse, le « user\_profil » représente l'ensemble des droits de l'utilisateur sur l'API alors que le « scope » représente un sous-ensemble de ces droits que l'utilisateur a consenti à déléguer à l'application consommatrice.

A noter que ces paramètres sont facultatifs et que leur utilisation sera limitée aux cas d'usage nécessitant un niveau de finesse important dans la gestion des habilitations utilisateurs.

**Dans les cas d'usage intra-Groupe, l'attribut « user\_profil » sera privilégié pour transmettre les habilitations fonctionnelles de l'utilisateur. Ensuite, l'attribut « scope » pourra être utilisé pour définir un sous-ensemble des habilitations fonctionnelles de l'utilisateur définies par son « user\_profil ». Le « scope » aura donc son utilité si parmi l'ensemble des droits de l'utilisateur, on ne souhaite déléguer qu'une sous partie de ces droits à l'application consommatrice pour l'accès à l'API.**

### 5.3.5.1.4 Synthèse

La mise en œuvre par les Entités (dans leur Authorization Server) des différentes cinématiques prévues dans la présente norme API pourra être progressive en fonction des besoins qui se présenteront.

Le tableau ci-dessous synthétise pour chaque cas d'usage les cinématiques envisageables (voir l'arbre de décisions précédemment décrit).

Cinématiques Natives  
OAuth2 / OIDC



<b>UC1-1</b>	ROPC Authorization Code
<b>UC1-2</b>	Client Credentials
<b>UC2-1</b>	ROPC Authorization Code
<b>UC2-2</b>	ROPC Authorization Code
<b>UC2-3</b>	ROPC Authorization Code
<b>UC2-4</b>	Client Credentials
<b>UC3-1</b>	ROPC Authorization Code

Les cinématiques détaillées correspondant à chaque cas d'usage sont détaillées dans les paragraphes suivants.

## 5.3.5.2 Client Credentials

### 5.3.5.2.1 Généralités

Cette cinématique est conçue pour l'authentification applicative : elle convient à la consommation de ressources ne nécessitant pas l'authentification préalable d'un utilisateur.

Avec cette cinématique, aucune information sur l'identité de l'utilisateur, éventuellement présentes dans la requête, ne peut être exploitée pour servir de preuve sur l'utilisateur à l'origine de la requête. En conséquence, aucun contrôle d'habilitation ou d'autorisation ne peut être fait en fonction de l'utilisateur (les habilitations/autorisations s'appuieront donc uniquement sur l'identifiant de l'application consommatrice) et aucune trace sur l'identité de l'utilisateur à l'origine de la requête n'est possible au niveau de l'API.

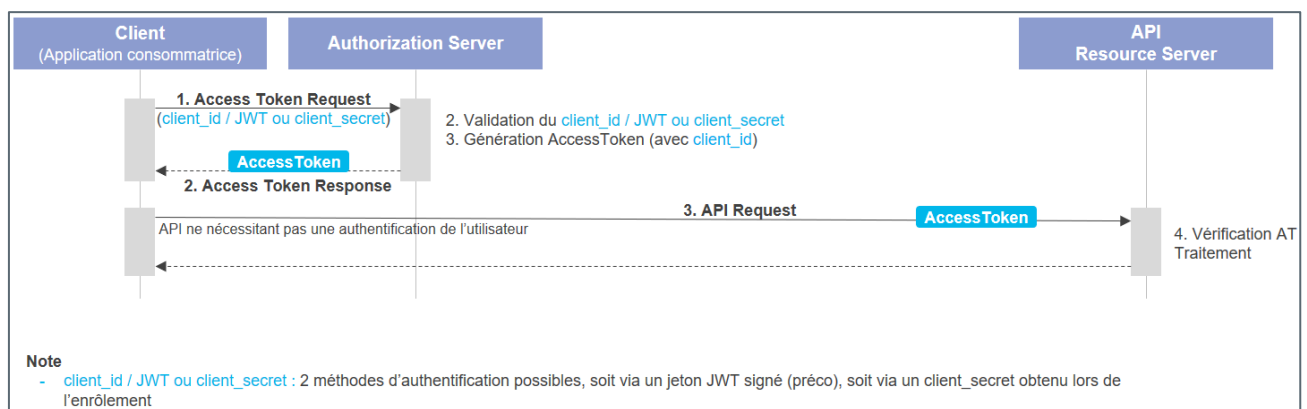
Ce mode Client Credentials n'est utilisable que si les ressources accédées n'appartiennent à aucun utilisateur et donc qu'aucune autorisation utilisateur n'est nécessaire.

A noter, qu'il faudra accorder un soin particulier au choix de la méthode d'authentification des applications consommatrices (voir §5.3.4). En effet, la sécurité de la cinématique Client Credentials s'appuie essentiellement sur la méthode d'authentification de l'application Client. Donc, si l'analyse de risque d'une API met en évidence sa sensibilité, l'authentification « private\_key\_jwt » pourra être mise en œuvre. Il s'agit d'authentifier l'application Client en s'appuyant sur un jeton JWT signée avec un certificat issu de la PKI Groupe (Carioca), plutôt que d'utiliser la méthode « client\_secret\_basic » (client\_id / client\_secret).

### 5.3.5.2.2 Cinématique générale et format des requêtes (Client Credentials Grant)

#### 5.3.5.2.2.1 Diagramme de séquence (Client Credentials Grant)

Cinématique générale Client Credentials (tout cas d'usage)



#### 5.3.5.2.2.2 Description détaillée des requêtes (Client Credentials Grant)

A noter que seules les paramètres obligatoires et les principaux paramètres optionnels sont repris dans les exemples de requête ci-dessous. Pour connaître l'exhaustivité des possibilités, il faut se référer aux [RFC des standards OAuth et OpenID Connect](#)

1. Access Token Request : description du contenu de la requête de demande d'Access Token pour la cinématique « Client Credentials » avec la méthode d'authentification « private\_key\_jwt », pour exemple, pour l'application consommatrice (Client) :

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_secret = MON_CLIENT_SECRET
&client_assertion_type = "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
&client_assertion = MON_CLIENT_JWT_TOKEN
&scope= MON_SCOPE //optionnel
```

Conformément au standard OAuth2 ([RFC6749](#)) et « JWT Profile for OAuth 2.0 Client Authentication » ([RFC7523](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre « grant\_type » → Obligatoire et égale à « client\_credentials »
- Paramètre « client\_assertion\_type » → Obligatoire si « private\_key\_jwt » ([§5.3.4.2](#))
  - Type de Jeton d'authentification de l'application consommatrice (Client).
  - Il sera toujours égale à : "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
- Paramètre « client\_assertion » → Obligatoire si « private\_key\_jwt » ([§5.3.4.2](#))
  - Jeton JWT d'authentification de l'application consommatrice.
  - Voir [§5.3.4.2](#)
- Paramètre « scope » → Optionnel

A noter que les paramètres « client\_assertion\_type » et « client\_assertion » seront remplacés par le paramètre « client\_secret » si la méthode d'authentification (de l'application Client) correspondante est choisie.

A noter, également, que la RFC8707 ([Resource Indicators for OAuth 2.0](#)) permet d'étendre l'Access Token Request en y ajoutant le paramètre http « resource » pour indiquer la liste des API (au format URI) pour lesquelles l'application Client souhaite un Access Token. Il existe également une autre extension du protocole OAuth ([OAuth 2.0: Audience Information](#)) qui propose d'utiliser le paramètre http « audience » pour répondre au même besoin.

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## 2. Access Token Response (Client Credentials Grant) :

Exemple de flux d'envoi d'un Access Token depuis l'Authorization Server vers l'application Cliente, en réponse à l'Access Token Request précédente.

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

{
  "access_token": "MON_ACCESS_TOKEN",
  "token_type": "bearer",
  "expires_in": DUREE_EN_SEC           //préconisé
}
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- TLS (HTTPS) → Obligatoire
- Header « Cache-Control: no-store » → Obligatoire
- Header « Pragma: no-cache » → Obligatoire
- Paramètre « access\_token » → Obligatoire
  - Cf. [5.3.5.4.5 Spécification de l'access token \(Authorization Code\)](#)
- Paramètre « token\_type » → Obligatoirement égale à « bearer »
- Paramètre « expires\_in » → Préconisé
  - Durée de vie (en secondes) de l'Access Token. Indiqué à titre indicatif pour permettre un accès simplifié à cette information qui se trouve également dans l'Access Token. Les contrôles de sécurité sur la validité du jeton doivent être effectués sur les dates d'expiration présentes dans l'Access Token et, donc, pas sur ce paramètre « expires\_in ».

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## 3. API Request :

Exemple de flux d'envoi d'un Access Token à une API.

```
GET /my_api HTTP/1.1
Host: api.ca-sa.group.gca
Authorization: Bearer MON_ACCESS_TOKEN
```

Conformément au standard OAuth2 ([RFC6749](#) et [RFC6750](#)), cette requête aura les caractéristiques suivantes :

- GET → Exemple

- TLS (HTTPS) → Obligatoire
- Paramètre Header « Authorization: Bearer » → Préconisé

A noter, que sous certaines conditions ([RFC6750](#)), il est possible de transmettre l'Access Token dans le corps de la requête (via un paramètre « access\_token »).

Si des données applicatives/fonctionnelles doivent être transmises à l'API, elles pourront être positionnées dans le corps de la requête. Si ces données doivent être signées et/ou chiffrées, il conviendra d'appliquer les recommandations du §5.2.2 (JWS, JWE, etc.). A noter que, parmi les données fonctionnelles/applicatives, il pourra y avoir les données d'habilitation de l'utilisateur décrites au [§5.1.3](#).

A noter, également, que dans le cas d'une utilisation d'une API Gateway en frontal de l'API, les données contenues ou associées à l'Access Token pourront être transmises à la ressource (i.e. backend) via la génération d'un nouveau jeton au format JWT. Par exemple, si l'API Gateway WSO2 est utilisée, le JWT sera transféré via le HEADER « X-JWT-Assertion ».


Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### Cinématique et description des réponses en erreur

[Cf. 5.3.5.5 Cinématique générale et format des réponses pour les erreurs](#)

#### 5.3.5.2.3 Mise en œuvre sur les cas d'usage

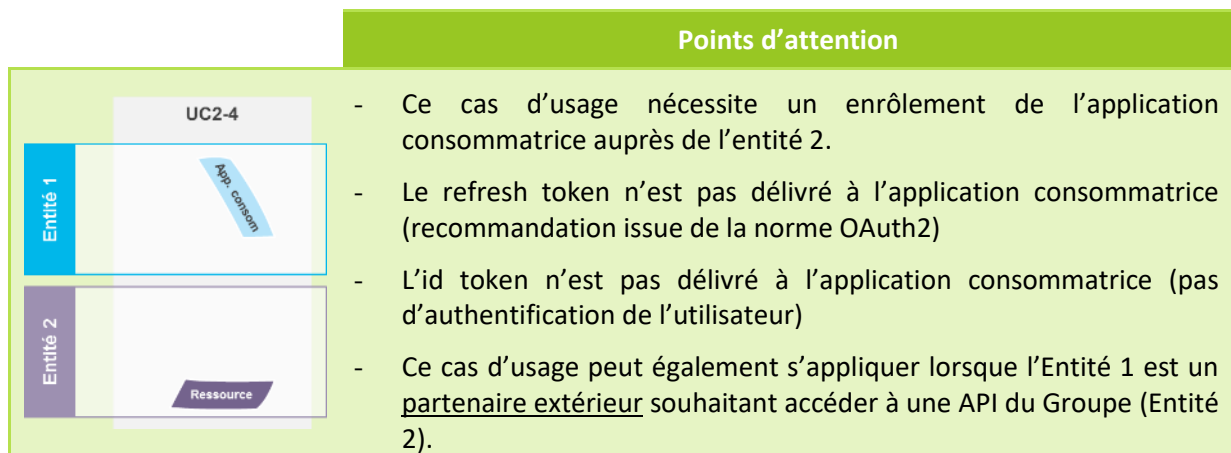
##### 5.3.5.2.3.1 UC1-2 Client Credentials

Points d'attention	
<div> <div>Entité 1</div> <div> <div>UC1-2</div>  </div> </div>	<ul style="list-style-type: none"> <li>- Le refresh token n'est pas délivré à l'application consommatrice (recommandation issue de la norme OAuth2)</li> <li>- L'id token n'est pas délivré à l'application consommatrice (pas d'authentification de l'utilisateur)</li> </ul>

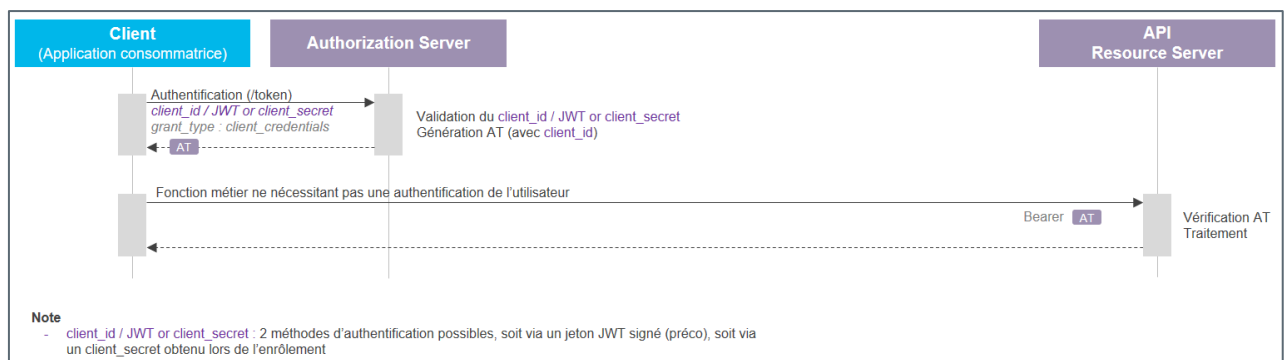
Le schéma ci-dessous illustre la cinématique associée à ce cas d'usage :



## 5.3.5.2.3.2 UC2-4 Client Credentials



Le schéma ci-dessous illustre la cinématique associée à ce cas d'usage :



## 5.3.5.2.4 Spécification de l'Access Token (Client Credentials)

### Format technique

**L'Access Token est de type Bearer et généré selon le format JSON Web Token (JWT) ou UUID (jeton dit opaque).**

A noter que le format JWT se distingue du jeton opaque par le fait qu'il est autoporteur. L'ensemble des données d'identification et de preuve d'authentification à destination de l'API et de la ressource sont présentes dans le jeton.

**Dans le cas du format UUID**, le jeton n'est qu'un simple identifiant UUID et ne porte donc aucune donnée. Les données associées à ce jeton restent stockées au niveau de l'Authorization Server et ainsi ne circulent pas sur le réseau et ne sont pas accessibles de l'application consommatrice. Ce format offre l'avantage de la légèreté du jeton (donc la performance) et de la sécurité, par contre, il n'est pas autoporteur (nécessite la récupération, par l'API, des données sur l'Authorization Server).

**Dans le cas du format JWT, le jeton est signé selon la spécification JSON Web Signature (JWS) par l'entité en charge de sa génération** : la signature est vérifiée par le fournisseur de l'API au moyen des données transmises dans l'en-tête du jeton.

**L'Access Token, au format JWT, peut être chiffré selon la spécification JSON Web Encryption par l'entité en charge de sa génération.** Cette disposition ne constitue pas une obligation, l'évaluation de la sensibilité des données qu'il contient est laissée à l'appréciation de l'entité. Voir chapitres 5.2.3 pour plus de détail.

## Structure de l'Access Token au format JWT pour Client Credentials

### Header

Intitulé	O/F	Usage	Description	Format
<b>alg</b>	O	Std. JWT/JWS	<b>Algorithme de signature</b> Recommandé : RS256	String
<b>typ</b>	F	Std. OAuth2	<b>media type du JWT</b> Recommandé : « at+JWT »	String
<b>cty</b>	O	Std. JWT/JWS	<b>Type de jeton</b> Valeur : JWT	String
<b>kid</b>	O	Std. JWT/JWS	<b>DN Certificat x509 utilisé pour la signature</b> Non renseigné si x5c, x5u ou x5t#S256 renseigné.	String
<b>x5c</b>		Std. JWT/JWS	<b>Certificat x509 utilisé pour la signature (généré par la PKI Groupe)</b> Non renseigné si kid, x5u ou x5t#S256 renseigné.	PEM
<b>x5u</b>		Std. JWT/JWS	<b>Lien vers le certificat x509 de signature</b> Non renseigné si kid, x5c ou x5t#S256 renseigné.	URI
<b>x5t#S256</b>		Std. JWT/JWS	<b>Empreinte sha-256 du certificat x509 de signature</b> Non renseigné si kid, x5c ou x5u renseigné.	String

Remarque : lorsque cela est possible (au regard des contraintes de performance selon le volume échangé et en fonction des possibilités de traitement à la réception du jeton), il est préconisé de privilégier l'alimentation de l'attribut x5c avec le certificat x509 pour avoir un jeton autoporteur et ainsi simplifier l'architecture et le processus de renouvellement du certificat.

### Body

Intitulé	O/F	Usage	Description	Format
<b>iss</b>	O	Std. OAuth2	<b>Émetteur du jeton</b> URI de l'autorization server	URI
<b>sub</b>	O	Std. OAuth2	<b>Identifiant du porteur</b>	String

aud	O	Std. OAuth2	<b>Audience(s) du jeton</b> Tableau de valeurs : [« identifiant API/ressource »] Cf. <a href="#">Glossaire des données utilisées dans les jetons</a>	[String or URI]
exp	O	Std. OAuth2	<b>Heure d'expiration du jeton</b>	Timestamp
client_id	O	Std. OAuth2	<b>Identifiant de l'application consommatrice</b>	String
iat	O	Std. OAuth2	<b>Heure de délivrance du jeton</b>	Timestamp
auth_time	O	Std. OAuth2	<b>Heure de l'authentification</b>	Timestamp
jti	O	Std. OAuth2	<b>Identifiant unique du jeton</b>	UUID
acr	O	Std. OAuth2	<b>Niveau de confiance de la preuve d'authentification</b> Valeur : Cf. chapitre 5.3.6	Integer
amr	F	Std. OAuth2	<b>Méthode d'authentification</b> Valeur : Cf. RFC8176	[String]
scope	O	Std. OAuth2	<b>Périmètre des droits (habilitations) délégués à l'application consommatrice, pour accéder à l'API</b>	String
Il est possible d'ajouter des données complémentaires en lien avec l'ident/authent, accessible par l'AS et nécessaire pour l'entité				
...				

Se référer au chapitre 5.3.5.6 pour plus de détail sur les attributs décrits ci-dessus.

**Données à mémoriser dans l'Authorization Server dans le cas d'un Access Token au format UUID, puis à transmettre à la ressource (backend), pour Client Credentials**

L'ensemble des données décrites dans le « body » du JWT (ci-dessus), doivent être mémorisées au niveau de l'Authorization Server pour pouvoir être transmises à la ressource (backend) lorsque le jeton opaque UUID sera reçu par l'API.



## 5.3.5.3 Resource Owner Password Credentials (ROPC)

### 5.3.5.3.1 Généralités

Cette cinématique convient à la consommation de ressources nécessitant l'authentification préalable d'un utilisateur. Elle ne peut être autorisée que pour une application consommatrice (Client) sûr dans la mesure où les informations de connexion (identifiant et mot de passe) de l'utilisateur transitent par ce dernier.

Il est préconisé de privilégier, dans la mesure du possible, la cinématique Authorization Code par rapport à la cinématique ROPC qui a plusieurs inconvénients :

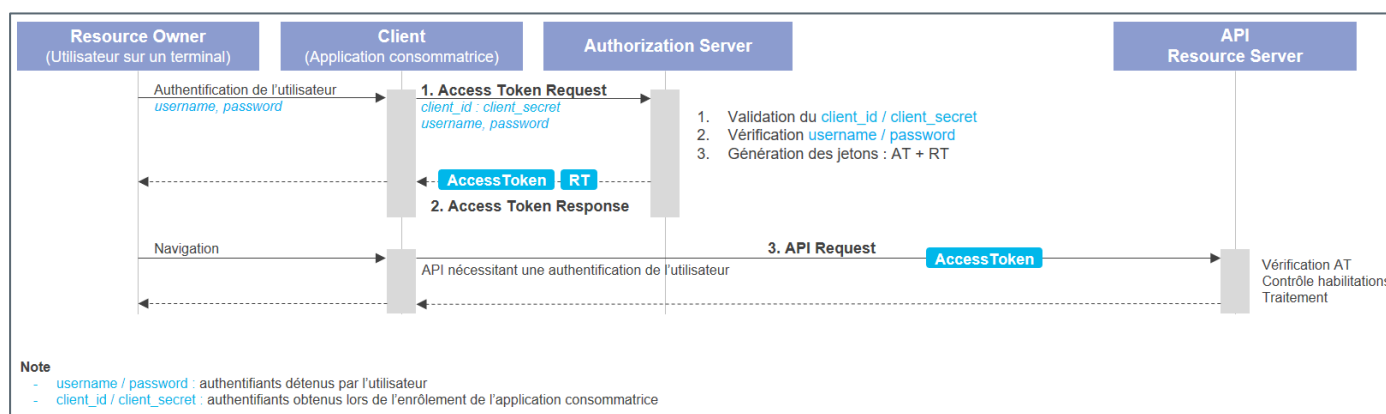
- Les identifiant/mot de passe de l'utilisateur transitent par l'application Cliente qui n'est pas toujours conçue pour le faire
- L'utilisateur ne valide pas les autorisations demandées par l'application Cliente (le consentement n'est pas demandé)
- Seule l'authentification par mot de passe est prévue dans le standard
- Le standard OpenID Connect ne l'interdit pas mais n'en détaille pas l'utilisation dans sa spécification.

A noter également que, comme indiqué au §5.3.4, les applications consommatrices d'API (« Client ») disposent de plusieurs méthodes pour s'authentifier sur l'Authorization Server. Il conviendra de choisir la méthode la mieux adaptée au contexte et, notamment, aux besoins de sécurité définis dans l'analyse de risque de l'application. Dans les requêtes ci-après, la méthode par défaut « client\_secret\_basic » (client\_id / client\_secret) est utilisée pour l'exemple.

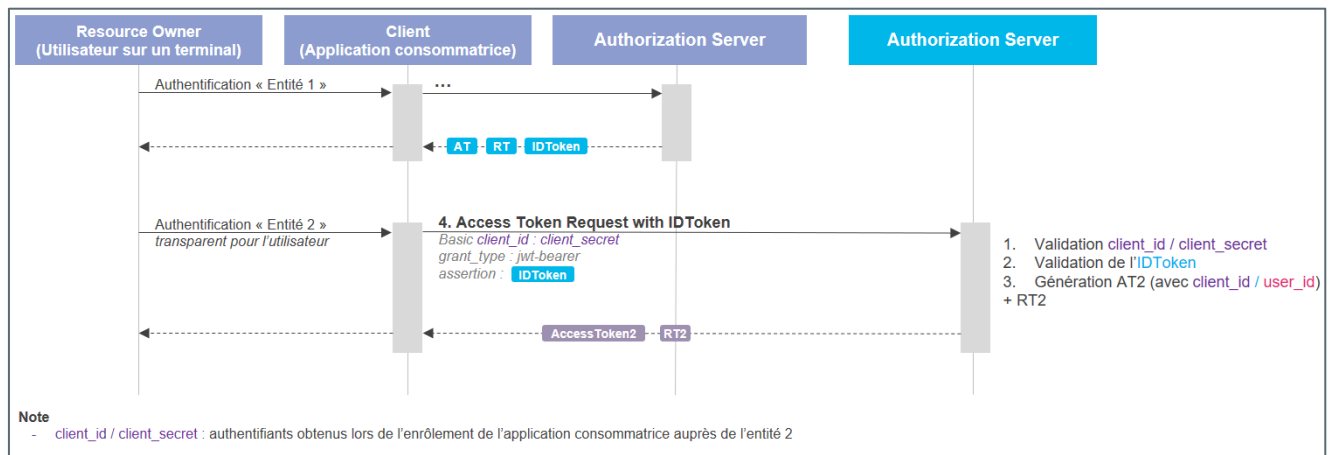
### 5.3.5.3.2 Cinématique générale et format des requêtes (ROPC Grant)

#### 5.3.5.3.2.1 Diagrammes de séquence (ROPC Grant)

##### Cinématique générale (tout cas d'usage)

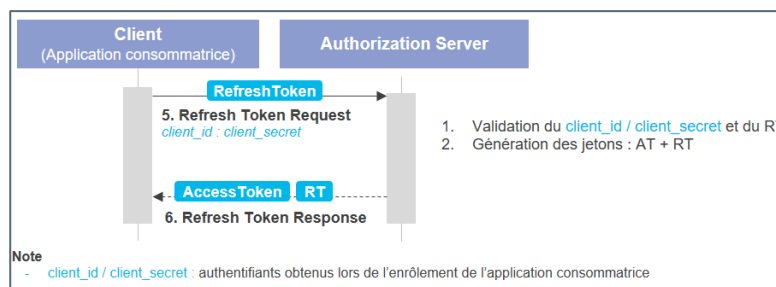


#### Cinématique de demande d'un Access Token à partir d'un ID Token JWT (tous cas d'usage multi-entité du Groupe)

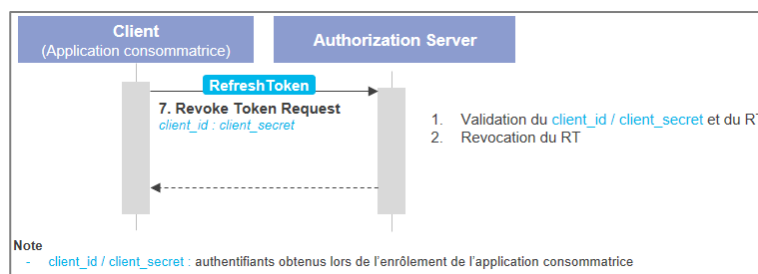


Remarque : si l'Authorization Server bleu est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

## Cinématique de renouvellement de l'Access Token (tout cas d'usage)



## Cinématique de révocation d'un Token (tout cas d'usage), par exemple le Refresh Token



### 5.3.5.3.2.2 Description détaillée des requêtes (ROPC Grant)

A noter que seules les paramètres obligatoires et les principaux paramètres optionnels sont repris dans les exemples de requêtes ci-dessous. Pour connaître l'exhaustivité des possibilités, il faut se référer aux [RFC des standards OAuth et OpenID Connect](#).

En préalable de cette cinématique, l'application Client se charge de récupérer l'identifiant et le mot de passe de l'utilisateur. Après récupération d'un Access Token, l'application devra supprimer toute référence au mot de passe de l'utilisateur (en mémoire ou autre stockage).

### 1. Access Token Request (ROPC Grant) :

Exemple de requête de demande d'Access Token entre l'Application Client et l'Authorization Server :

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type=password
&username=LOGIN_UTILISATEUR
&password=PASSWORD_UTILISATEUR
&scope=MON_SCOPE //optionnel
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
  
- Paramètre Header « Authorization: Basic » → Obligatoire
  - o Utilisation recommandée de l'HTTP Basic Auth pour transmettre les client\_id et client\_secret de l'application.
  - o Une autre solution possible consiste à transmettre les paramètres « client\_id » et « client\_secret » dans le corps de la requête.
  - o A noter que d'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).
  
- Paramètre « grant\_type » → Obligatoire et égale à « password »
- Paramètre « username » → Obligatoire
- Paramètre « password » → Obligatoire
- Paramètre « scope » → Optionnel
  - o Doit contenir la valeur « openid » pour obtenir un IDToken en réponse de la requête de demande de jeton

A noter, également, que la RFC8707 ([Resource Indicators for OAuth 2.0](#)) permet d'étendre l'Access Token Request en y ajoutant le paramètre http « resource » pour indiquer la liste des API (au format URI) pour lesquelles l'application Client souhaite un Access Token. Il existe également une autre extension du protocole OAuth ([OAuth 2.0: Audience Information](#)) qui propose d'utiliser le paramètre http « audience » pour répondre au même besoin.

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## 2. Access Token Response (ROPC Grant) :

Exemple de flux d'envoi d'un Access Token depuis l'Authorization Server vers l'application Client, en réponse à l'Access Token Request.

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

{
  "access_token": "MON_ACCESS_TOKEN",
  "token_type": "bearer",
  "expires_in": DUREE_EN_SEC,           //préconisé
  "refresh_token": "MON_REFRESH_TOKEN", //optionnel
  "id_token": "MON_ID_TOKEN"           //optionnel
}
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- TLS (HTTPS) → Obligatoire
- Header « Cache-Control: no-store » → Obligatoire
- Header « Pragma: no-cache » → Obligatoire
- Paramètre « access\_token » → Obligatoire
  - Cf. [5.3.5.4.4 Spécification de l'access token \(Authorization Code\)](#)
- Paramètre « token\_type » → Obligatoirement égale à « bearer »
- Paramètre « refresh\_token » → Optionnel
- Paramètre « id\_token » → Optionnel
  - Mais obligatoirement présent si la valeur « openid » était présente dans le scope transmis par l'application consommatrice lors de l'Authorization Request
- Paramètre « expires\_in » → Préconisé
  - Durée de vie (en secondes) de l'Access Token. Indiqué à titre indicatif pour permettre un accès simplifié à cette information qui se trouve également dans l'Access Token. Les contrôles de sécurité sur la validité du jeton doivent être effectués sur les dates d'expiration présentes dans l'Access Token et, donc, pas sur ce paramètre « expires\_in ».

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## 3. API Request :

Cf. l'API Request décrite au [§5.3.5.2.2 Cinématique générale et format des requêtes \(Client Credentials Grant\)](#)

#### 4. Access Token Request with ID Token :

Cette requête est décrite par la [RFC7523](#) (« Using JWTs as Authorization Grants »).

Exemple de requête de demande d'Access Token entre l'Application Client et l'Authorization Server en fournissant un IDToken comme preuve d'authentification de l'utilisateur :

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=ID_TOKEN
&scope=MON_SCOPE //optionnel
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre Header « Authorization: Basic » → Obligatoire
  - L'HTTP Basic Auth est la solution la plus courante pour transmettre les client\_id et client\_secret de l'application (mais aussi la moins sécurisée. D'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).
- Paramètre « grant\_type » → Obligatoire et égale à « urn:ietf:params:oauth:grant-type:jwt-bearer » au format URL Encoded
  - Sauf si l'Authorization Server cible de la requête est un Keycloak. Dans ce cas, le grant\_type « token-exchange » sera privilégié (Voir la remarque ci-après).
- Paramètre « assertion » → Obligatoire
  - Contient le jeton JWT (IDToken) constituant la preuve d'authentification de l'utilisateur
- Paramètre « scope » → Optionnel
  - Doit contenir la valeur « openid » pour obtenir un IDToken en réponse de la requête de demande de jeton

Remarque 1 : si l'Authorization Server cible de la requête est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la [RFC8693](#) (OAuth 2.0 Token Exchange) et donc utiliser les paramètres suivants pour transmettre l'ID Token :

- Paramètre « grant\_type » → Obligatoire et égale à « urn:ietf:params:oauth:grant-type:token-exchange » au format URL Encoded
- Paramètre « subject\_token » → Obligatoire
  - Contient le jeton JWT (IDToken) constituant la preuve d'authentification de l'utilisateur
- Paramètre « subject\_token\_type » → Obligatoire et égale à « urn:ietf:params:oauth:token-type:idtoken » au format URL Encoded

Remarque 2 : la RFC8707 ([Resource Indicators for OAuth 2.0](#)) permet d'étendre l'Access Token Request en y ajoutant le paramètre http « resource » pour indiquer la liste des API (au format URI) pour lesquelles l'application Client souhaite un Access Token. Il existe également une autre extension du protocole OAuth ([OAuth 2.0: Audience Information](#)) qui propose d'utiliser le paramètre http « audience » pour répondre au même besoin.

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### 5. Refresh Token Request (ROPC Grant) :

Exemple de requête de demande de renouvellement d'un Access Token entre l'Application Client et l'Authorization Server :

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type= refresh_token
&refresh_token= MON_REFRESH_TOKEN
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre Header « Authorization: Basic » → Obligatoire
  - Utilisation recommandée de l'HTTP Basic Auth pour transmettre les client\_id et client\_secret de l'application.

- Une autre solution possible consiste à transmettre les paramètres « client\_id » et « client\_secret » dans le corps de la requête.
- A noter que d'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).

- Paramètre « grant\_type » → Obligatoire et égale à « refresh\_token »
- Paramètre « refresh\_token » → Obligatoire

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### 6. Refresh Token Response (ROPC Grant) :

Identique à « Access Token Response (ROPC Grant) »

### 7. Revoke Token Request (ROPC Grant) :

Exemple de requête de révocation du Refresh Token entre l'Application Cliente et l'Authorization Server :

```
POST /revoke HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

token= MON_TOKEN
token_type_hint=refresh_token //optionnel
```

Conformément au standard OAuth2 ([RFC7009](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre Header « Authorization: Basic » → Obligatoire
  - Utilisation recommandée de l'HTTP Basic Auth pour transmettre les client\_id et client\_secret de l'application.
  - Une autre solution possible consiste à transmettre les paramètres « client\_id » et « client\_secret » dans le corps de la requête.
  - A noter que d'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).
- Paramètre « token » → Obligatoire
  - Si le token est un refresh\_token, alors tous les access\_token associés (du même Authorization Grant) seront également révoqués

- Paramètre « token\_type\_hint » → Optionnel

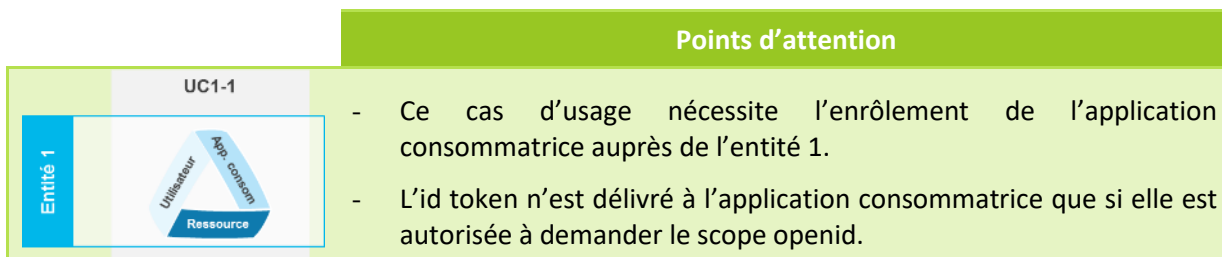
Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## Cinématique et description des réponses en erreur

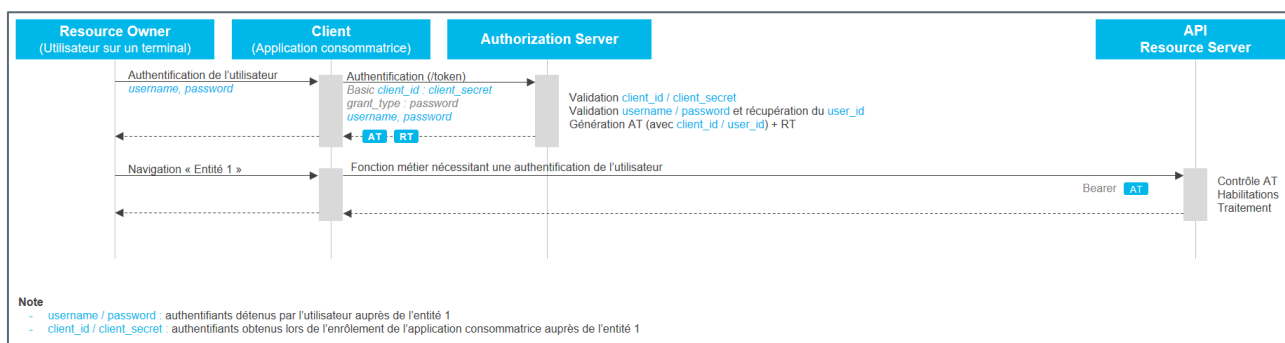
[Cf. 5.3.5.5 Cinématique générale et format des réponses pour les erreurs](#)

### 5.3.5.3.3 Application sur les cas d'usage

#### 5.3.5.3.3.1 UC1-1 ROPC



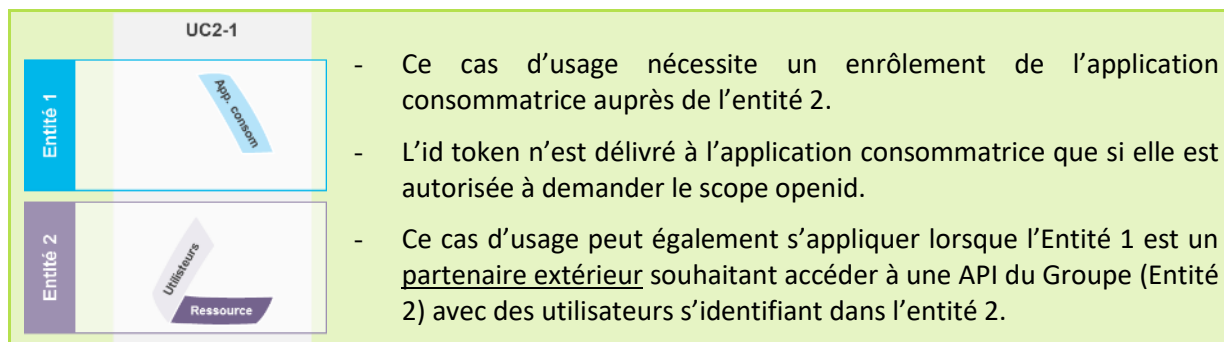
Le schéma ci-dessous illustre la cinématique d'authentification associée à ce cas d'usage :



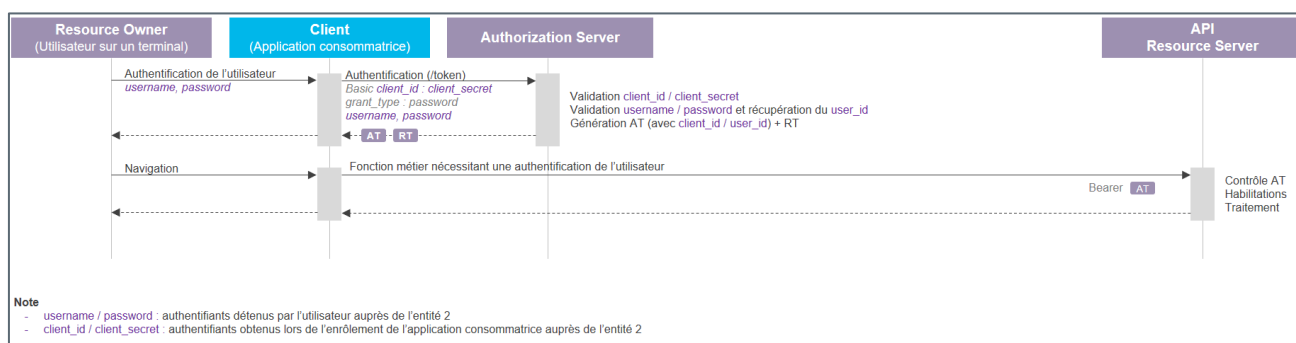
#### 5.3.5.3.3.2 UC2-1 ROPC





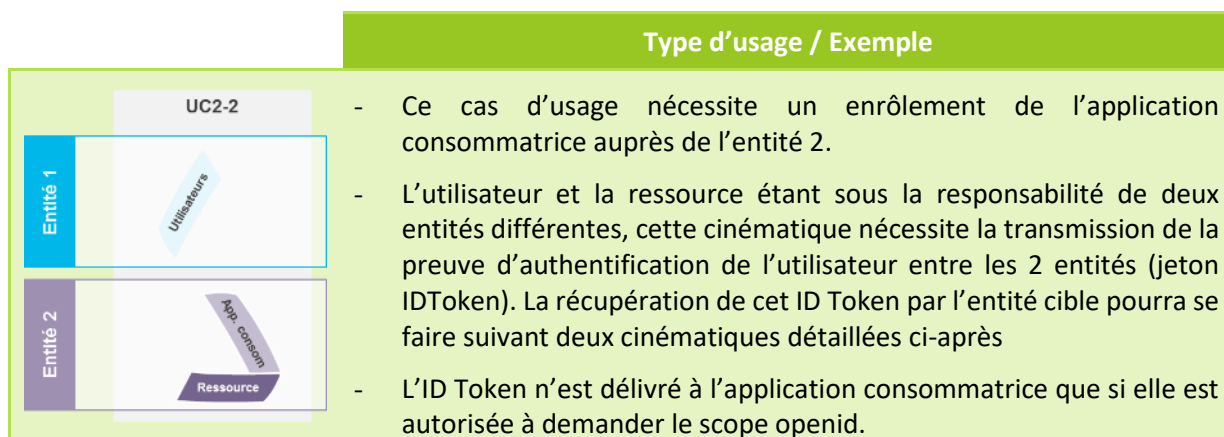


Le schéma ci-dessous illustre la cinématique d'authentification associée à ce cas d'usage :



Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#).

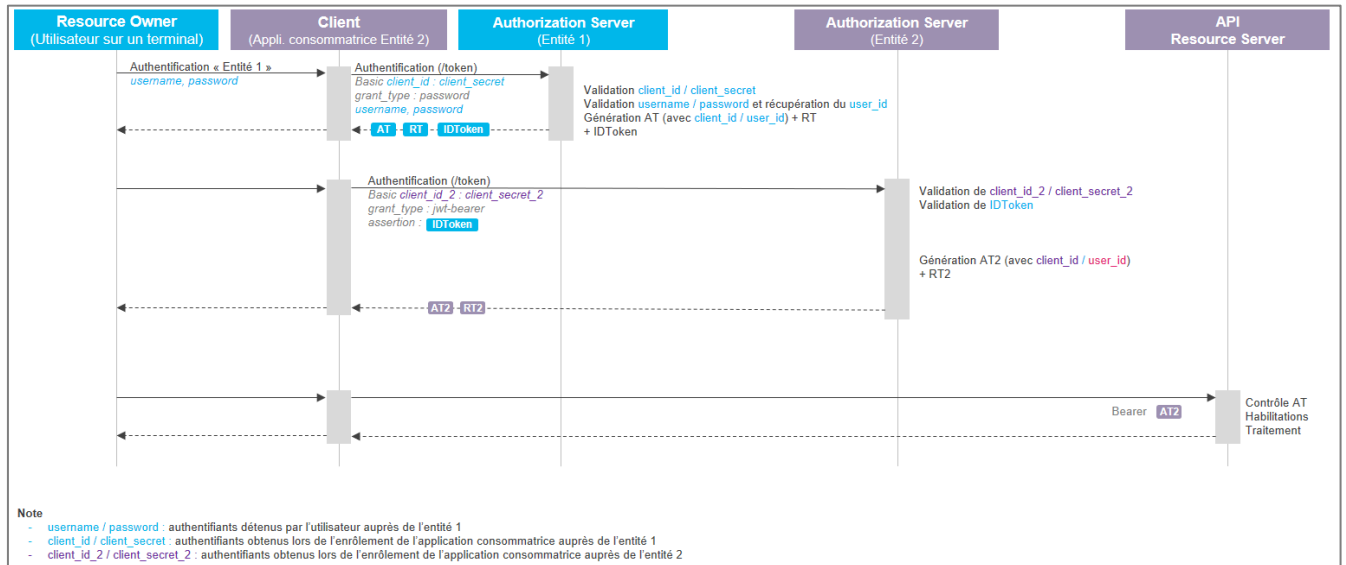
## 5.3.5.3.3 UC2-2 ROPC



Dans cette cinématique, l'ID Token reçu par l'application consommatrice (Client) est également transmis à l'Authorization Server de l'entité fournissant l'API. Cette ID Token produite par l'Entité 1 contiendra

donc dans son attribut « audience » les deux applicatifs destinés à le recevoir (application Client et AZ Server).

Le schéma ci-dessous illustre cette 1<sup>ère</sup> cinématique d'authentification associée à ce cas d'usage :



A noter qu'en cas d'utilisation de la cinématique ROPC, l'application Client ne recevra pas de cookie de session en réponse à la requête d'authentification (/token) et donc seule l'ID token reçu pourra servir de preuve d'authentification de l'utilisateur et permettre le SSO avec l'Authorization Server de l'entité 2. Et donc, la cinématique standard de SSO via OpenID Connect tel que décrit pour le flow AZ code au §5.3.5.4.4.3.2, ne sera pas applicable avec ROPC.

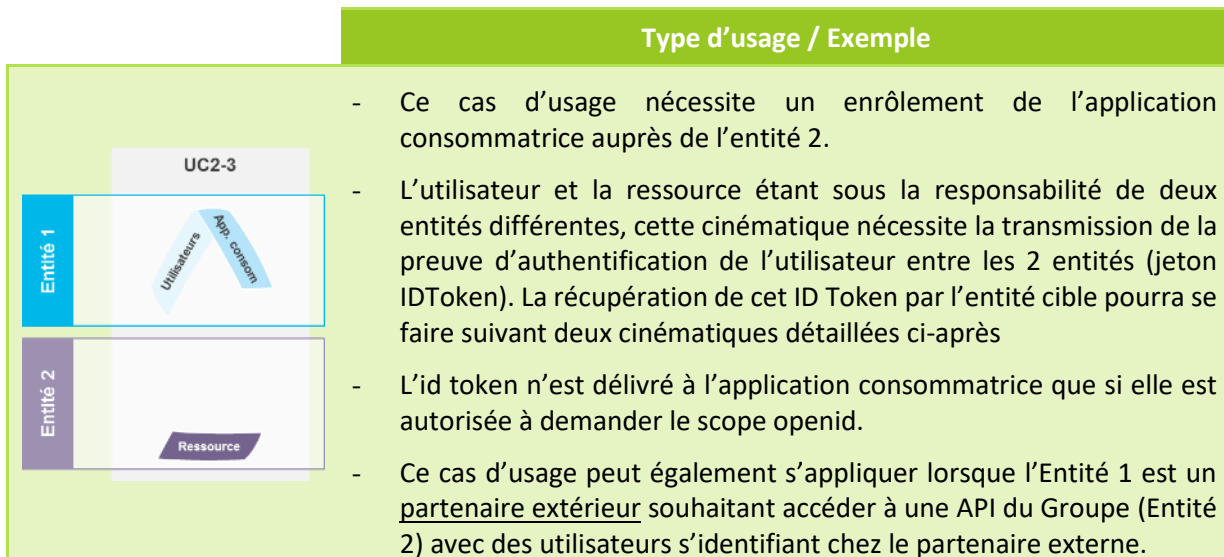
Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

Si l'application consommatrice est accédée via NNI IHM ou autre SSO, cf. [5.5.2 Cas des applications compatibles NNI IHM](#)

## Autres remarques :

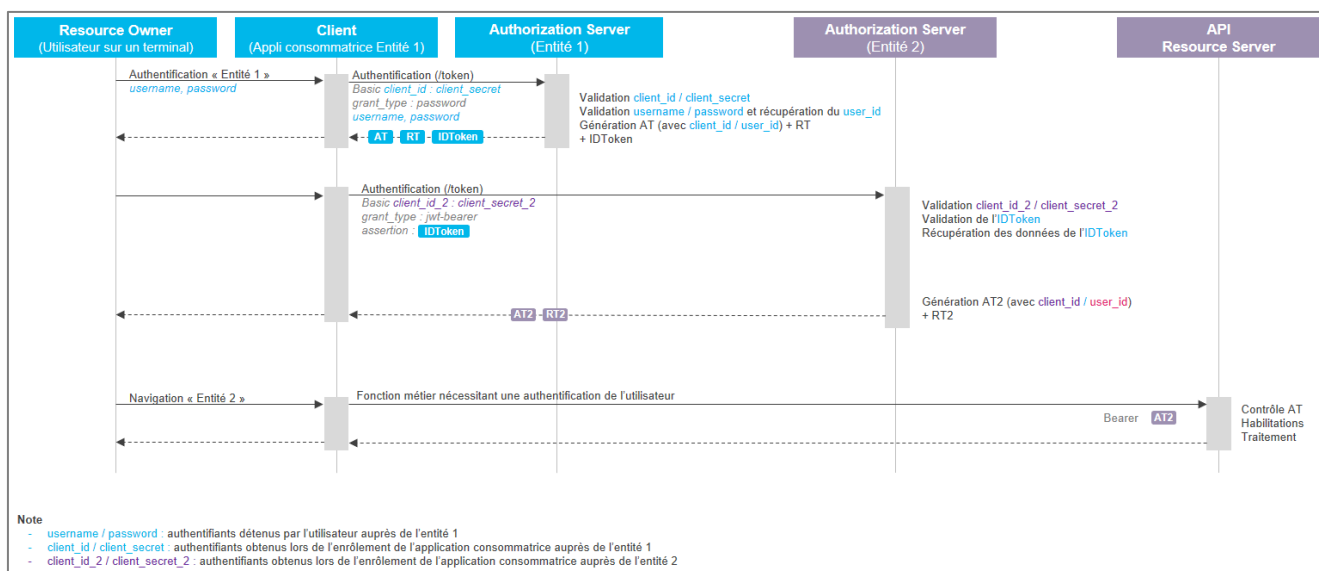
- Le jeton IDToken n'est utilisé que pour initialiser la session d'authentification de l'utilisateur sur l'autorization server de l'Entité cible (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect). Ainsi, la durée de validité du jeton IDToken ne sera que de quelques minutes.
- Après réception du jeton IDToken par l'application consommatrice, la génération des jetons d'accès à l'API est de la responsabilité de l'Entité hébergeant l'API et l'application. La cinématique OAuth/OIDC proposée ici côté Entité cible n'est qu'une bonne pratique (non normatif).
- Si l'Authorization Server violet de l'entité 2 est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

## 5.3.5.3.3.4 UC2-3 ROPC



Dans cette cinématique, l'ID Token reçu par l'application consommatrice (Client) est également transmis à l'Authorization Server de l'entité fournissant l'API. Cette ID Token produite par l'Entité 1 contiendra dans son attribut « audience » les deux applicatifs destinés à le recevoir (application Client et AZ Server).

Le schéma ci-dessous illustre cette 1<sup>ère</sup> cinématique d'authentification associée à ce cas d'usage :



A noter qu'en cas d'utilisation de la cinématique ROPC, l'application Client ne recevra pas de cookie de session en réponse à la requête d'authentification (/token) et donc seule l'ID token reçu pourra servir de preuve d'authentification de l'utilisateur et permettre le SSO avec l'Authorization Server de l'entité

2. Et donc, la cinématique standard de SSO via OpenID Connect tel que décrit pour le flow AZ code au §5.3.5.4.4.2, ne sera pas applicable avec ROPC.

Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

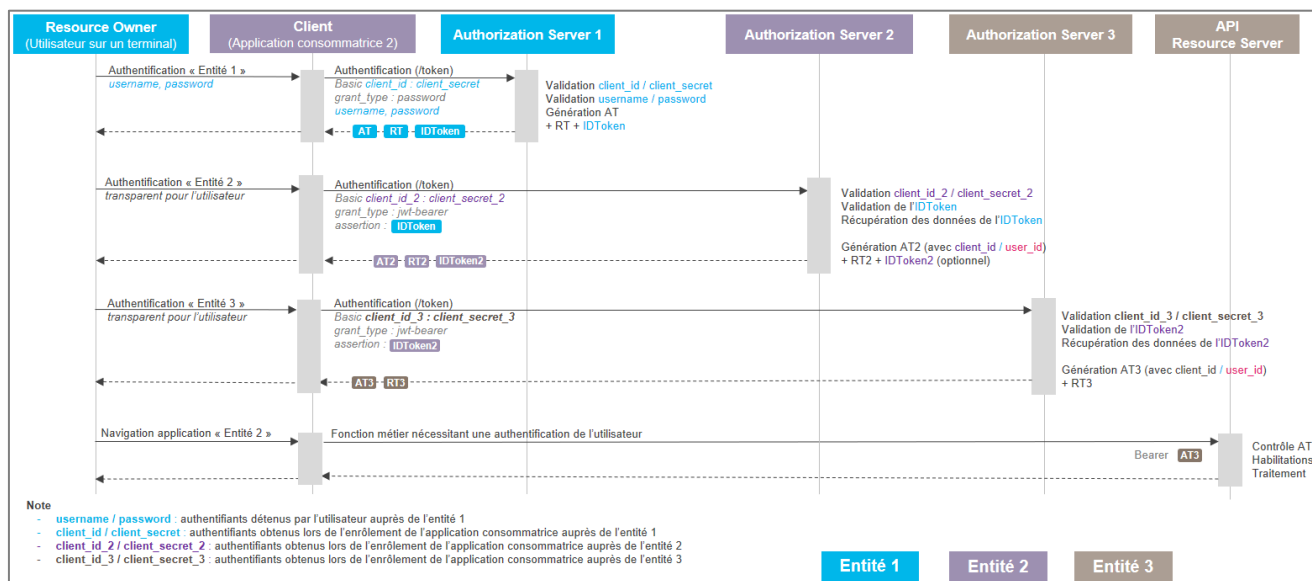
Le jeton IDToken n'est utilisé que pour initialiser la session d'authentification de l'utilisateur sur l'autorization server de l'Entité cible (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect). Ainsi, la durée de validité du jeton IDToken ne sera que de quelques minutes.

A noter, également, que si l'Authorization Server violet de l'entité 2 est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

### 5.3.5.3.5 UC3-1 ROPC

		Type d'usage / Exemple
UC3-1		
Entité 1	Utilisateur	<ul style="list-style-type: none"> <li>- Ce cas d'usage nécessite un enrôlement de l'application consommatrice auprès de l'entité 3.</li> <li>- L'utilisateur, l'application et la ressource étant sous la responsabilité de trois entités différentes, cette cinématique nécessite la transmission de la preuve d'authentification de l'utilisateur entre les 3 entités (jetons IDToken)</li> <li>- L'id token n'est délivré à l'application consommatrice que si elle est autorisée à demander le scope openid.</li> </ul>
Entité 2	App. consom.	
Entité 3	Ressource	

Le schéma ci-dessous illustre la cinématique associée à ce cas d'usage :



Comme pour les cinématiques UC2-2 ROPC et UC2-3 ROPC, il est possible de mettre en œuvre une cinématique alternative dans laquelle les AZ Server 2 et 3 joueraient le rôle de broker d'identité en redirigeant l'utilisateur vers son identity provider (l'AZ Server 1) pour obtenir un ID Token.

Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

Si l'application consommatrice est accédée via NNI IHM ou autre SSO, cf. [5.5.2 Cas des applications compatibles NNI IHM](#)

A noter, également, que si les Authorization Server 2 ou 3 sont des Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

## 5.3.5.3.4 Spécification de l'access token (ROPC)

### Format technique

**L'access token est de type Bearer et généré selon le format JSON Web Token (JWT) ou UUID (jeton dit opaque).**

A noter que le format JWT se distingue du jeton opaque par le fait qu'il est autoporteur. L'ensemble des données à destination de l'API et de la ressource sont présentes dans le jeton.

**Dans le cas du format UUID,** le jeton n'est qu'un simple identifiant UUID et ne porte donc aucune donnée. Les données associées à ce jeton restent stockées au niveau de l'Authorization Server et ainsi ne circulent pas sur le réseau et ne sont pas accessibles de l'application consommatrice. Ce format

offre l'avantage de la légèreté du jeton (donc la performance) et de la sécurité, par contre, il n'est pas autoporteur (nécessite la récupération, par l'API, des données sur l'Authorization Server).

**Il est signé selon la spécification JSON Web Signature (JWS) par l'entité en charge de sa génération :** la signature est vérifiée par le fournisseur de l'API au moyen des données transmises dans l'en-tête du jeton.

**L'access token, au format JWT, peut être chiffré selon la spécification JSON Web Encryption par l'entité en charge de sa génération.** Cette disposition ne constitue pas une obligation, l'évaluation de la sensibilité des données qu'il contient est laissée à l'appréciation de l'entité. Voir chapitres 5.2.3 et 5.3.3 pour plus de détail.

## Structure de l'Access Token au format JWT pour ROPC

### Header

Intitulé	O/F	Usage	Description	Format
alg	O	Std. JWT/JWS	<b>Algorithme de signature</b> Recommandé : « RS256 »	String
typ	F	Std. OAuth2	<b>media type du JWT</b> Recommandé : « at+JWT »	String
cty	O	Std. JWT/JWS	<b>Type de jeton</b> Valeur : JWT	String
kid	O	Std. JWT/JWS	<b>DN Certificat x509 utilisé pour la signature</b> Non renseigné si x5c, x5u ou x5t#S256 renseigné.	String
x5c		Std. JWT/JWS	<b>Certificat x509 utilisé pour la signature (généré par la PKI Groupe)</b> Non renseigné si kid, x5u ou x5t#S256 renseigné.	PEM
x5u		Std. JWT/JWS	<b>Lien vers le certificat x509 de signature</b> Non renseigné si kid, x5c ou x5t#S256 renseigné.	URI
x5t#S256		Std. JWT/JWS	<b>Empreinte certificat x509 de signature</b> Non renseigné si kid, x5c ou x5u renseigné.	String

Remarque : lorsque cela est possible (au regard des contraintes de performance selon le volume échangé et en fonction des possibilités de traitement à la réception du jeton), il est préconisé de privilégier l'alimentation de l'attribut x5c avec le certificat x509 pour avoir un jeton autoporteur et ainsi simplifier l'architecture et le processus de renouvellement du certificat.

### Body

Intitulé	O/F	Usage	Description	Format
iss	O	Std. OAuth2	<b>Émetteur du jeton</b> URI de l'Authorization server	URI
sub	O	Std. OAuth2	<b>Identifiant du porteur</b>	String
aud	O	Std. OAuth2	<b>Audience(s) du jeton</b> Tableau de valeurs : [« identifiant API/ressource »] Cf. <a href="#">Glossaire des données utilisées dans les jetons</a>	[String or URI]
exp	O	Std. OAuth2	<b>Heure d'expiration du jeton</b>	Timestamp
client_id	O	Std. OAuth2	<b>Identifiant de l'application consommatrice</b>	String
iat	O	Std. OAuth2	<b>Heure de délivrance du jeton</b>	Timestamp
auth_time	O	Std. OAuth2	<b>Heure de l'authentification</b>	Timestamp
jti	O	Std. OAuth2	<b>Identifiant unique du jeton</b>	UUID
acr	O	Std. OAuth2	<b>Niveau de confiance de la preuve d'authentification</b> Valeur : Cf. chapitre 5.3.6	Integer
amr	F	Std. OAuth2	<b>Méthode d'authentification</b> Valeur : Cf. RFC8176	[String]

Scope	O	Std. OAuth2	Périmètre des droits (habilitations) délégués à l'application consommatrice, pour accéder à l'API	String
authorization_context	O		Contexte d'autorisation	JSON Object
session_id	F	Habilitations Traçabilité	Identifiant de la session d'authentification initiale de l'utilisateur final Il s'agit d'un identifiant généré lors de l'authentification initiale de l'utilisateur final et associé à sa session d'authentification initiale (sur le portail Distributeur ou l'IDP associé, par exemple)	UUID
user_type	O	Traçabilité	Type d'utilisateur	String
user_uom_code	O	Habilitations Traçabilité	UO Métier d'appartenance de l'utilisateur	String
user_id	O	Habilitations Traçabilité	Identifiant de l'utilisateur	String
Il est possible d'ajouter des données complémentaires en lien avec l'ident/authent, accessible par l'AS et nécessaire pour l'entité				
...				

Se référer au chapitre 5.3.5.6 pour plus de détail sur les attributs décrits ci-dessus.

**Données à mémoriser dans l'Authorization Server dans le cas d'un Access Token au format UUID, puis à transmettre à la ressource (backend), pour ROPC**

L'ensemble des données décrites dans le « body » du JWT ci-dessus, doivent être mémorisées au niveau de l'Authorization Server pour pouvoir être transmises à la ressource (backend) lorsqu'un jeton opaque UUID sera reçu par l'API.

## 5.3.5.3.5 Spécification de l'ID Token

### Principes et description générale

L'ID Token est généré si le scope « openid » est demandé par l'application cliente à l'Authorization Server.

Pour la description générale de l'ID Token : Cf. [5.3.3 Types de Jetons](#)

### Format technique

**L'ID Token est au format JSON Web Token (JWT).**

**Il est signé selon la spécification JSON Web Signature (JWS) par l'entité en charge de sa génération.**  
La signature est vérifiée par le destinataire de l'ID Token au moyen des données transmises dans l'en-tête du jeton.

### Structure de l'id token ROPC

#### Header

Intitulé	O/F	Usage	Description	Format
alg	O	Std. JWT/JWS	<b>Algorithme de signature</b> Recommandé : RS256	String
cty	O	Std. JWT/JWS	<b>Type de jeton</b> Valeur : JWT	String
kid	O	Std. JWT/JWS	<b>DN Certificat x509 utilisé pour la signature</b> Non renseigné si x5c, x5u ou x5t#S256 renseigné.	String
x5c		Std. JWT/JWS	<b>Certificat x509 utilisé pour la signature (généré par la PKI Groupe)</b> Non renseigné si kid, x5u ou x5t#S256 renseigné.	PEM
x5u		Std. JWT/JWS	<b>Lien vers le certificat x509 de signature</b> Non renseigné si kid, x5c ou x5t#S256 renseigné.	URI
x5t#S256		Std. JWT/JWS	<b>Empreinte certificat x509 de signature</b> Non renseigné si kid, x5c ou x5u renseigné.	String

Remarque : lorsque l'ID token est issue d'un Identity Provider de type OpenID Provider, le standard OpenID Connect prévoit que celui-ci mette à disposition un « endpoint » HTTP pour récupérer le certificat de signature de l'ID Token (au format JWKS). Ainsi, la solution, couramment constatée sur le marché, consiste à ne pas transmettre le certificat complet dans l'ID Token (donc, ne pas renseigner l'attribut x5c) mais plutôt de transmettre l'attribut kid ou x5u ou x5t#S256. Charge ensuite au destinataire du jeton ID Token d'aller rechercher le certificat à l'URL mise à disposition par l'Identity Provider émetteur du jeton.

## Body

- Scope « openid » :

Le scope « openid » est le scope standard pour mettre en œuvre OpenId Connect et obtenir un IDToken.

**Les entités doivent fournir les informations minimum suivantes lorsque le scope « openid » est demandé par l'application consommatrice (sous réserve qu'elle y soit autorisée).**

## Body

Intitulé	O/F	Usage	Description	Format
iss	O	Std. OAuth2	<b>Émetteur du jeton</b> Case Sensitive URL de l'autorization server ayant produit le jeton.	URL
sub	O	Std. OAuth2	<b>Identifiant du porteur</b>	String
aud	O	Std. OAuth2	<b>Audience(s) du jeton</b> Tableau de valeurs : [« identifiant API ou ressource ou application »] Contient les identifiants des applicatifs autorisés à recevoir le jeton. Cf. <a href="#">Glossaire des données utilisées dans les jetons</a>	[String or URI]
exp	O	Std. OAuth2	<b>Heure d'expiration du jeton</b>	Timestamp
iat	O	Std. OAuth2	<b>Heure de délivrance du jeton</b>	Timestamp
auth_time	O	Std. OAuth2	<b>Heure de l'authentification</b>	Timestamp
nonce	F	Std. OpenID Connect	<b>Identifiant transmis par l'application Client lors de la requête de demande de jeton.</b>	String
jti	F	Std. JWT	<b>Identifiant unique du jeton</b>	String
acr	O	Std. OAuth2	<b>Niveau de confiance de la preuve d'authentification</b> Valeur : Cf. chapitre 5.3.6	Integer
amr	F	Std. OAuth2	<b>Méthode d'authentification</b> Valeur : Cf. RFC8176	[String]



Remarque : lorsque l'ID token est réceptionné par un WSO2, il faudra que l'identifiant de l'Identity Provider déclaré dans WSO2 soit égal à l'attribut « iss » présent dans l'ID Token reçu. De plus, il faudra, que l'alias de cet Identity Provider déclaré dans WSO2 soit présent dans l'attribut « aud » (audience) de l'ID Token reçu.

Se référer au chapitre 5.3.5.6 pour plus de détail sur les attributs décrits ci-dessus.

- Scope « user-id » :

**En plus du scope « openid », la norme API a prévu la possibilité pour l'application consommatrice de demander le scope « user-id ». Si l'application est bien habilitée à ce scope, l'IDToken produit devra ajouter les informations minimum suivantes (dans le body de l'IDToken)**

Body

Intitulé	O/F	Usage	Description	Format
authorization_context	O		Contexte d'autorisation	JSON Object
user_type	O	Traçabilité	Type d'utilisateur	String
user_uom_code	O	Habilitations Traçabilité	UO Métier d'appartenance de l'utilisateur	String
user_id	O	Habilitations Traçabilité	Identifiant de l'utilisateur	String

Se référer au chapitre 5.3.5.6 pour plus de détail sur les attributs décrits ci-dessus.

- Scope « user-ca » :

**En plus du scope « openid », la norme API a prévu la possibilité pour l'application consommatrice de demander le scope « user-ca ». Si l'application est bien habilitée à ce scope, l'IDToken produit devra ajouter les informations minimum suivantes (dans le body de l'IDToken après les données du scope openid)**

Body

Intitulé	O/F	Usage	Description	Format
authorization_context	O		Contexte d'autorisation	JSON Object
session_id	F	Habilitations Traçabilité	Identifiant de la session d'authentification initiale de l'utilisateur final	UUID
client_id	O	Traçabilité	Identifiant de l'application consommatrice	String
user_type	O	Traçabilité	Type d'utilisateur	String
user_uom_code	O	Habilitations Traçabilité	UO Métier d'appartenance de l'utilisateur	String
user_id	O	Habilitations Traçabilité	Identifiant de l'utilisateur	String
Possibilité d'ajouter des données complémentaires en lien avec l'ident/authent, accessible par l'AS et nécessaire pour l'entité				
...				

Se référer au chapitre 5.3.5.6 pour plus de détail sur les attributs décrits ci-dessus.

- Autres Scopes possibles :

**Dans le cadre de la génération de l'ID Token, les entités sont libres d'intégrer des scopes complémentaires en fonction des besoins. Il peut s'agir de scopes :**

1. Standards définis par OpenID Connect
2. Spécifiques en utilisant les possibilités d'extension de la norme OpenID Connect

#### **5.3.5.4 Authorization Code**

##### **5.3.5.4.1 Généralités**

Cette cinématique est conçue pour l'authentification d'un utilisateur : elle convient à la consommation de ressources nécessitant l'authentification préalable d'un utilisateur. Elle est à privilégier par rapport à Resource Owner Password Credentials dans la mesure où elle permet :

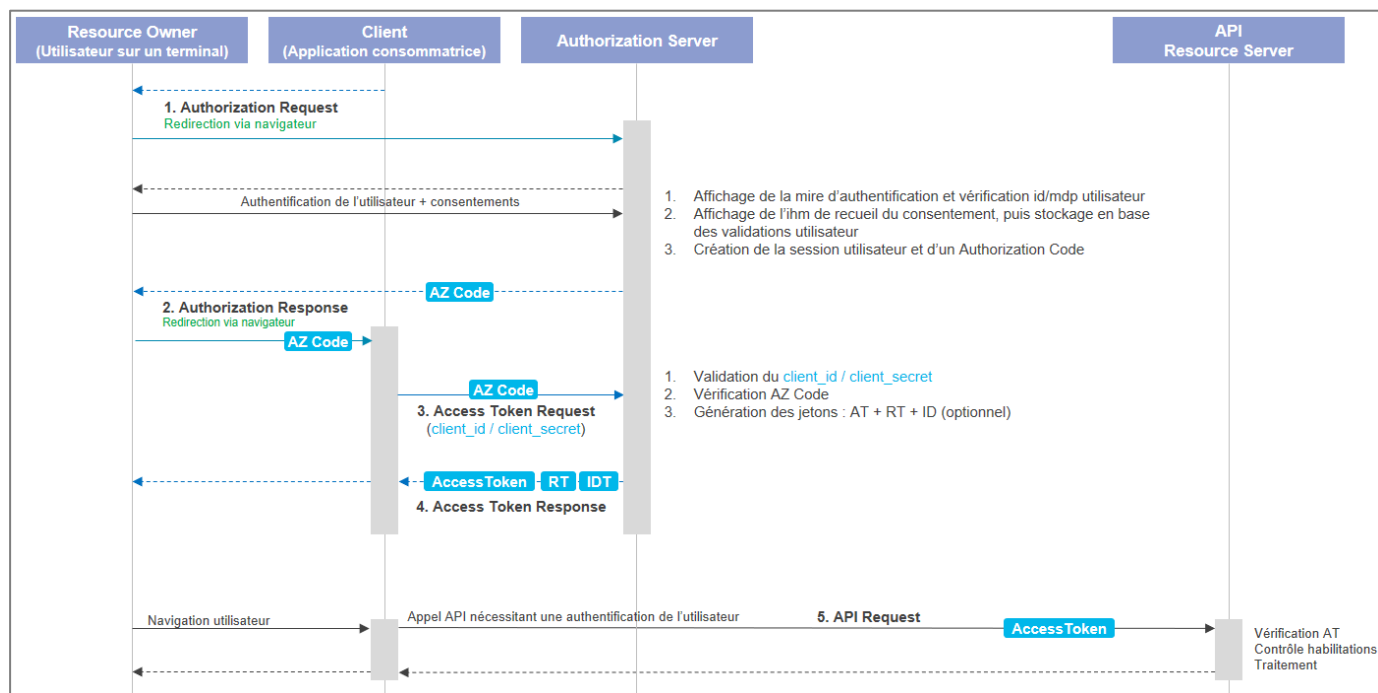
- De renforcer le niveau de sécurité en évitant de faire transiter le mot de passe utilisateur par l'application consommatrice
- De simplifier la gestion de l'authentification utilisateur en la déportant sur une interface centralisée accessible en multicanal
- De profiter d'une fonctionnalité SSO (Single Sign-On) via cette interface d'authentification

A noter également que, comme indiqué au [§5.3.4](#), les applications consommatrices d'API (« Client ») disposent de plusieurs méthodes pour s'authentifier sur l'Authorization Server. Il conviendra de choisir la méthode la mieux adaptée au contexte et, notamment, aux besoins de sécurité définis dans l'analyse de risque de l'application. Dans les requêtes ci-après, la méthode par défaut « client\_secret\_basic » (client\_id / client\_secret) est utilisée pour l'exemple.

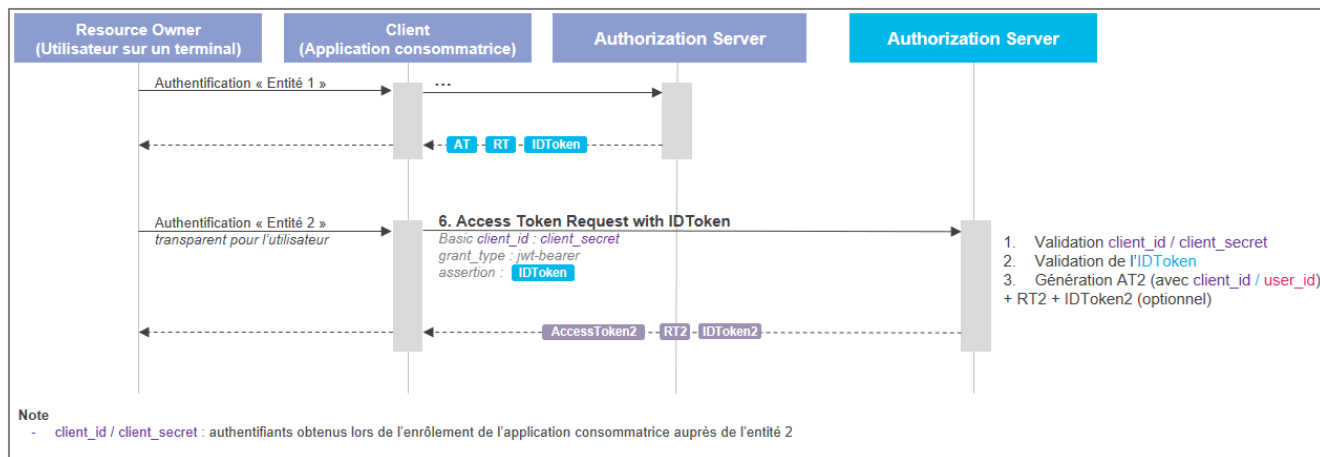
##### **5.3.5.4.2 Cinématique générale et format des requêtes (Authorization Code Grant)**

###### **5.3.5.4.2.1 Diagrammes de séquence (Authorization Code Grant)**

Cinématique générale (tout cas d'usage)

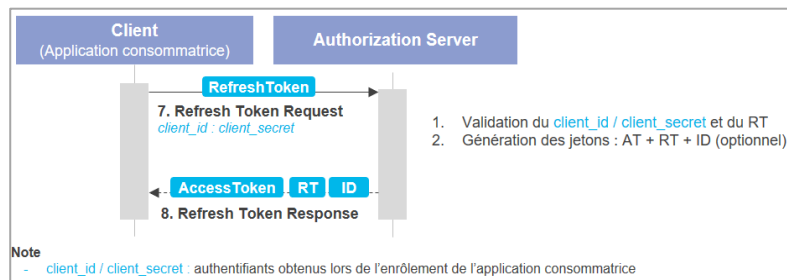


## Cinématique de demande d'un Access Token à partir d'un ID Token JWT (cas d'usage multi-entité du Groupe)

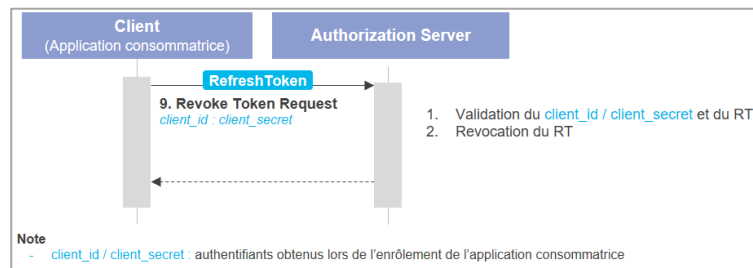


**Remarque :** si l'Authorization Server bleu est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier un utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

## Cinématique de renouvellement de l'Access Token (tout cas d'usage)



## Cinématique de révocation d'un Token (tout cas d'usage), par exemple le Refresh Token



### 5.3.5.4.2.2 Description détaillée des requêtes (AZ Code Grant)

A noter que seules les paramètres obligatoires et les principaux paramètres optionnels sont repris dans les exemples de requêtes ci-dessous (d'autres paramètres optionnels existent dans les standards OAuth/OIDC).

#### 1. Authorization Request (Authorization Code Grant) :

```

GET /authorize?
    response_type=code
    &client_id= MON_CLIENT_ID
    &state= MON_STATE_ALEA
    &scope= MON_SCOPE // optionnel
    &redirect_uri=MON_URL_APPLi // optionnel
    &nonce= MON_NONCE // optionnel
    &code_challenge=PKCE_CODE_CHALLENGE // optionnel (PKCE)
    &code_challenge_method=MA_PKCE_METHOD // optionnel (PKCE)
HTTP/1.1

Host: api.ca-sa.group.gca
    
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- GET → Bonne pratique
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre « reponse\_type » → Obligatoire et avec la valeur « code »
- Paramètre « client\_id » → Obligatoire
- Paramètre « state » → Obligatoire

- Paramètre « scope » → Optionnel
  - Doit contenir la valeur « openid » pour obtenir un IDToken en réponse de la requête de demande de jeton (cf. requête 4 : Access Token Response)
- Paramètre « redirect\_uri » → Optionnel
- Paramètre « nonce » → Optionnel
- Paramètre « code\_challenge » → Optionnel
  - Si l'application consommatrice est publique (Public Client), mise en œuvre de PKCE (cf. § 5.3.5.4.3)
- Paramètre « code\_challenge\_method » → Optionnel
  - Si l'application consommatrice est publique (Public Client), mise en œuvre de PKCE (cf. § 5.3.5.4.3)

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### 2. Authorization Response (Authorization Code Grant) :

La réponse à la requête d'autorisation sera une redirection vers l'application Client. Il s'agit d'un flux de l'Authorization Server vers le navigateur de l'utilisateur

```
HTTP/1.1 302 Found
Location: MON_URL_APPLi?
                    code=MON_AUTHORIZATION_CODE
                    &state=MON_STATE_ALEA
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- GET → Bonne pratique
- TLS (HTTPS) → Obligatoire
- Redirection HTTP → Bonne pratique (d'autres formes de redirection sont possibles)
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre « code » → Obligatoire
- Paramètre « state » → Obligatoire si présent dans la requête d'autorisation

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### 3. Access Token Request (Authorization Code Grant) :

Exemple de requête de demande d'Access Token entre l'Application Client et l'Authorization Server

```
POST /token HTTP/1.1
```

```
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type=      authorization_code
&code=           MON_AUTHORIZATION_CODE
&redirect_uri=    MON_URL_APPLi           //optionnel
&code_verifier=   MON_CODE_VERIFIER       //optionnel (PKCE)
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre Header « Authorization: Basic » → Obligatoire
  - Utilisation recommandée de l'HTTP Basic Auth pour transmettre les client\_id et client\_secret de l'application.
  - Une autre solution possible consiste à transmettre les paramètres « client\_id » et « client\_secret » dans le corps de la requête.
  - A noter que d'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).
- Paramètre « grant\_type » → Obligatoire et égale à « authorization\_code »
- Paramètre « code » → Obligatoire
- Paramètre « redirect\_uri » → Optionnel
  - Mais obligatoire si déjà présent dans la requête d'autorisation, sinon optionnel. Si présent, l'Authorization Server devra vérifier que l'URL de redirection envoyée ici est bien identique à l'URL de redirection initialement transmise lors de la requête précédente d'autorisation.
  -
- Paramètre « code\_verifier » → Optionnel
  - Si l'application consommatrice est publique (Public Client), mise en œuvre de PKCE (cf. [§ 5.3.5.4.3](#))

A noter, également, que la RFC8707 ([Resource Indicators for OAuth 2.0](#)) permet d'étendre l'Access Token Request en y ajoutant le paramètre http « resource » pour indiquer la liste des API (au format URI) pour lesquelles l'application Client souhaite un Access Token. Il existe également une autre extension du protocole OAuth ([OAuth 2.0: Audience Information](#)) qui propose d'utiliser le paramètre http « audience » pour répondre au même besoin.

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

#### 4. Access Token Response (Authorization Code Grant) :

Exemple de flux d'envoi d'un Access Token depuis l'Authorization Server vers l'application Client.

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

{
  "access_token": "MON_ACCESS_TOKEN",
  "token_type": "bearer",
  "expires_in": DUREE_EN_SEC, //préconisé
  "refresh_token": "MON_REFRESH_TOKEN", //optionnel
  "id_token": "MON_ID_TOKEN" //optionnel
}
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- TLS (HTTPS) → Obligatoire
- Header « Cache-Control: no-store » → Obligatoire
- Header « Pragma: no-cache » → Obligatoire
- Paramètre « access\_token » → Obligatoire
  - Cf. [5.3.5.4.4 Spécification de l'access token \(Authorization Code\)](#)
- Paramètre « token\_type » → Obligatoirement égale à « bearer »
- Paramètre « refresh\_token » → Optionnel
- Paramètre « id\_token » → Optionnel
  - Mais obligatoirement présent si la valeur « openid » était présente dans le scope transmis par l'application consommatrice lors de l'Authorization Request
- Paramètre « expires\_in » → Préconisé
  - Durée de vie (en secondes) de l'Access Token. Indiqué à titre indicatif pour permettre un accès simplifié à cette information qui se trouve également dans l'Access Token. Les contrôles de sécurité sur la validité du jeton doivent être effectués sur les dates d'expiration présentes dans l'Access Token et, donc, pas sur ce paramètre « expires\_in ».

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

## 5. API Request :

Cf. l'API Request décrite au [§5.3.5.2.2 Cinématique générale et format des requêtes \(Client Credentials Grant\)](#)

## 6. Access Token Request with ID Token :

Cette requête est décrite par la [RFC7523](#).

Exemple de requête de demande d'Access Token entre l'Application Client et l'Authorization Server en fournissant un IDToken comme preuve d'authentification de l'utilisateur :

```
POST /token HTTP/1.1
Host: api.ca-sa.group.gca
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Base64 (MON_CLIENT_ID:MON_CLIENT_SECRET)

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=ID_TOKEN
&scope=MON_SCOPE //optionnel
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- POST → Obligatoire
- TLS (HTTPS) → Obligatoire
- Format « application/x-www-form-urlencoded » → Obligatoire
- Paramètre Header « Authorization: Basic » → Obligatoire
  - Utilisation recommandée de l'HTTP Basic Auth pour transmettre les client\_id et client\_secret de l'application.
  - Une autre solution possible consiste à transmettre les paramètres « client\_id » et « client\_secret » dans le corps de la requête.
  - A noter que d'autres méthodes d'authentification de l'application consommatrice (Client) peuvent être utilisées. Elles sont décrites au [§5.3.4](#).
- Paramètre « grant\_type » → Obligatoire et égale à « urn:ietf:params:oauth:grant-type:jwt-bearer » au format URL Encoded
  - Sauf si l'Authorization Server cible de la requête est un Keycloak. Dans ce cas, le grant\_type « token-exchange » sera privilégié (Voir la remarque ci-après)
- Paramètre « assertion » → Obligatoire
  - Contient le jeton JWT (IDToken) constituant la preuve d'authentification de l'utilisateur
- Paramètre « scope » → Optionnel
  - Doit contenir la valeur « openid » pour obtenir un IDToken en réponse de la requête de demande de jeton

Remarque 1 : si l'Authorization Server cible de la requête est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, il faudra transmettre l'ID Token via la [RFC8693](#) (OAuth 2.0 Token Exchange) et donc utiliser les paramètres suivants pour transmettre l'ID Token :



- Paramètre « grant\_type » → Obligatoire et égale à « urn:ietf:params:oauth:grant-type:token-exchange » au format URL Encoded
- Paramètre « subject\_token » → Obligatoire
  - Contient le jeton JWT (IDToken) constituant la preuve d'authentification de l'utilisateur
- Paramètre « subject\_token\_type » → Obligatoire et égale à « urn:ietf:params:oauth:token-type:id\_token » au format URL Encoded

Remarque 2 : la RFC8707 ([Resource Indicators for OAuth 2.0](#)) permet d'étendre l'Access Token Request en y ajoutant le paramètre http « resource » pour indiquer la liste des API (au format URI) pour lesquelles l'application Client souhaite un Access Token. Il existe également une autre extension du protocole OAuth ([OAuth 2.0: Audience Information](#)) qui propose d'utiliser le paramètre http « audience » pour répondre au même besoin.

Pour une description détaillée de ces paramètres : Cf. [5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC](#)

### 7. Refresh Token Request (Authorization Code Grant) :

Identique à « Refresh Token Request (ROPC Grant) » : cf. [5.3.5.3.2 Cinématique générale et format des requêtes \(ROPC Grant\)](#)

### 8. Refresh Token Response (Authorization Code Grant) :

Identique à « Refresh Token Response (ROPC Grant) » : cf. [5.3.5.3.2 Cinématique générale et format des requêtes \(ROPC Grant\)](#)

### 9. Revoke Token Request (Authorization Code Grant) :

Identique à « Revoke Token Request (ROPC Grant) » : cf. [5.3.5.3.2 Cinématique générale et format des requêtes \(ROPC Grant\)](#)

### Cinématique et description des réponses en erreur

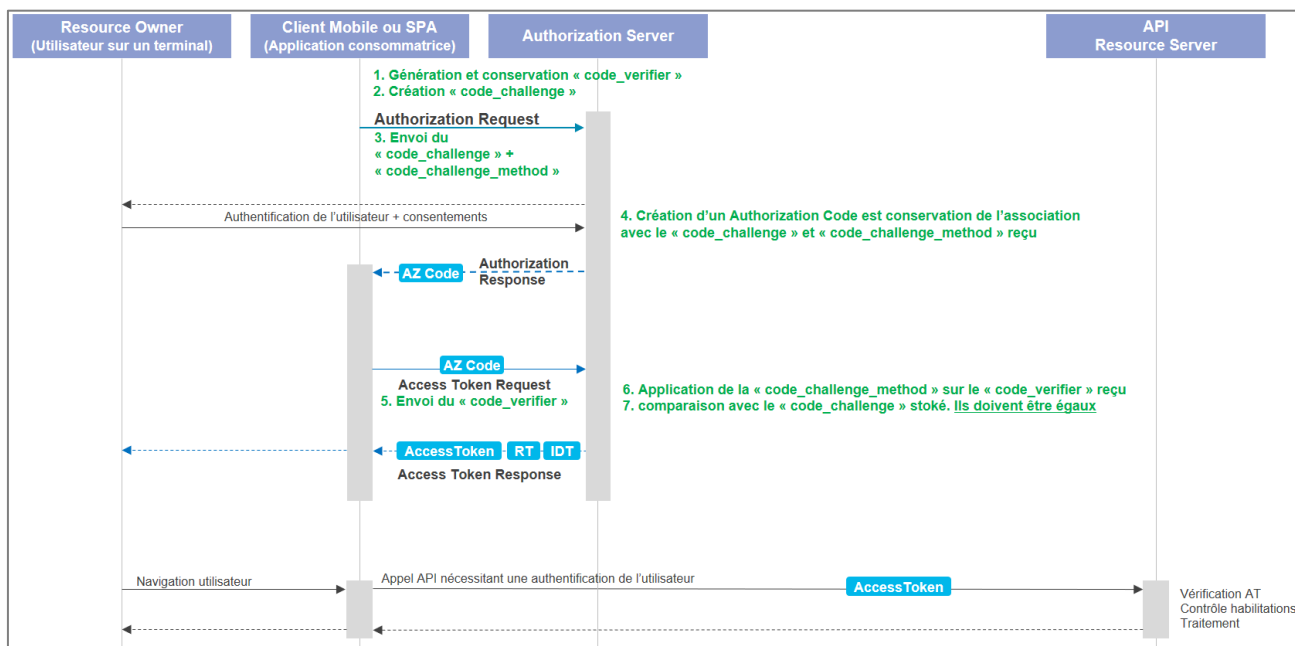
[Cf. 5.3.5.5 Cinématique générale et format des réponses pour les erreurs](#)

## 5.3.5.4.3 Cas particulier des applications consommatrices publiques (Public Clients)

Les applications consommatrices publiques (Public Clients) sont définies par OAuth comme étant des applications qui ne sont pas en mesure de garder confidentiel leur client\_secret (comme par exemple les applications mobiles ayant un client\_secret embarqué).

Pour ces applications, lorsque l'Authorization Code Grant est utilisé, il est nécessaire (cf. [MESARI API](#)) de mettre en œuvre la sécurisation PKCE (cf. [RFC7636](#)).

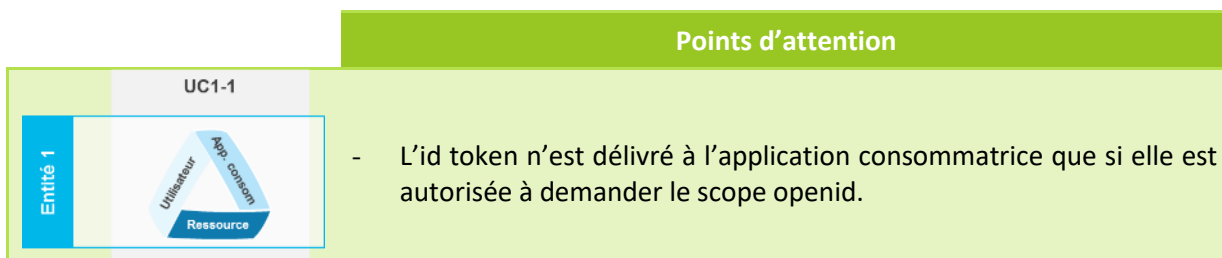
La mise en œuvre de la sécurisation PKCE dans le cadre d'une application mobile dans une cinématique AZ Code, est la suivante :



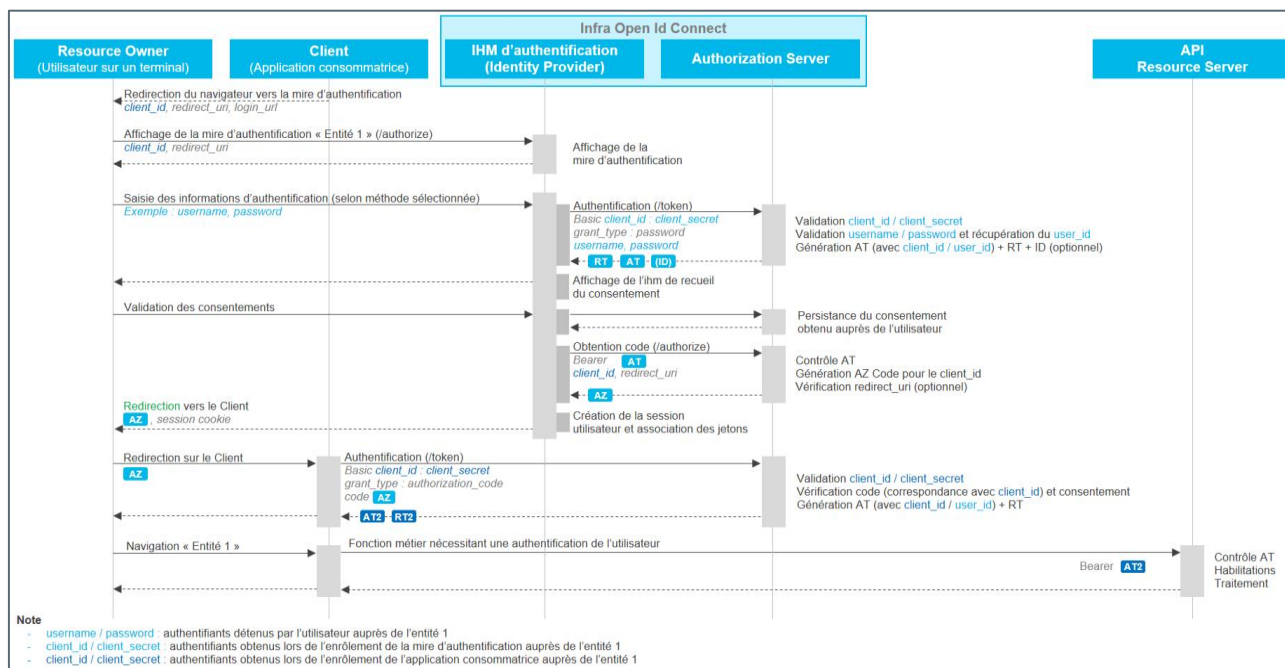
Cf. la [RFC7636](#) pour plus de détails.

## 5.3.5.4.4 Application sur les cas d'usage

### 5.3.5.4.4.1 UC1-1 Authorization Code

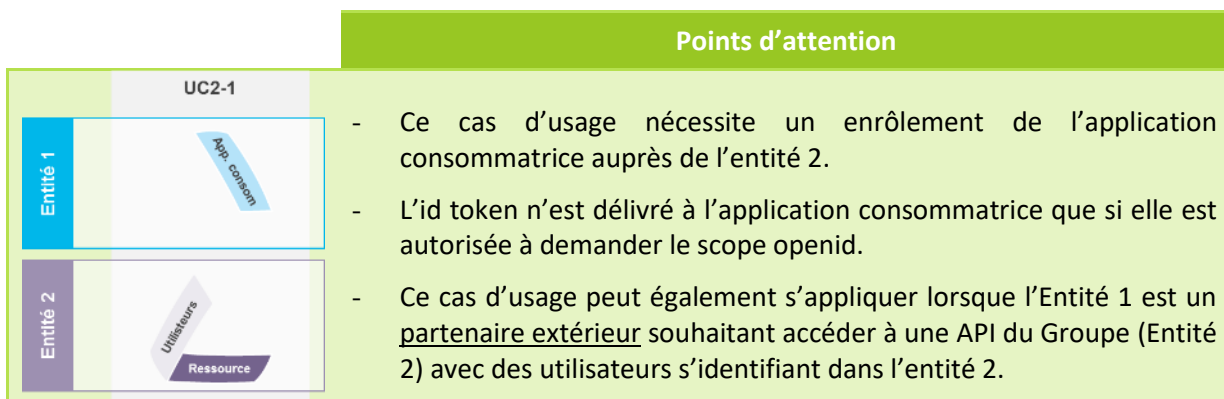


Le schéma ci-dessous illustre la cinématique d'authentification associée à ce cas d'usage :

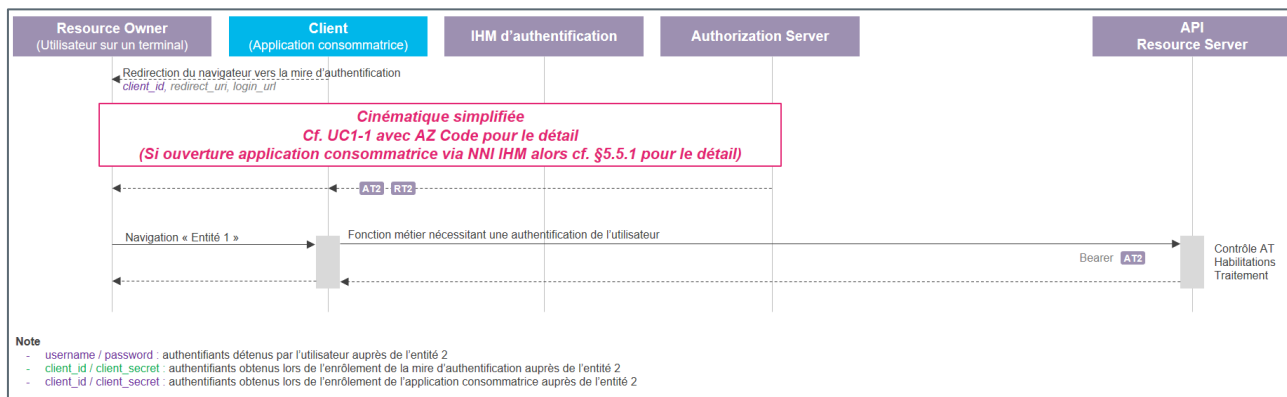


Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#).

## 5.3.5.4.4.2 UC2-1 Authorization Code



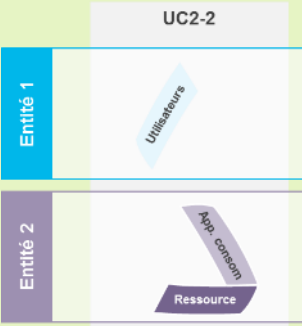
Le schéma ci-dessous illustre la cinématique d'authentification associée à ce cas d'usage :



La cinématique OpenID Connect mise en œuvre (correspondant au rectangle rouge du diagramme) peut soit être la même que UC1-1 (sans NNI IHM), soit être celle décrite au paragraphe [5.5.1](#) dans le cas d'une ouverture de l'application avec NNI IHM.

Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#).

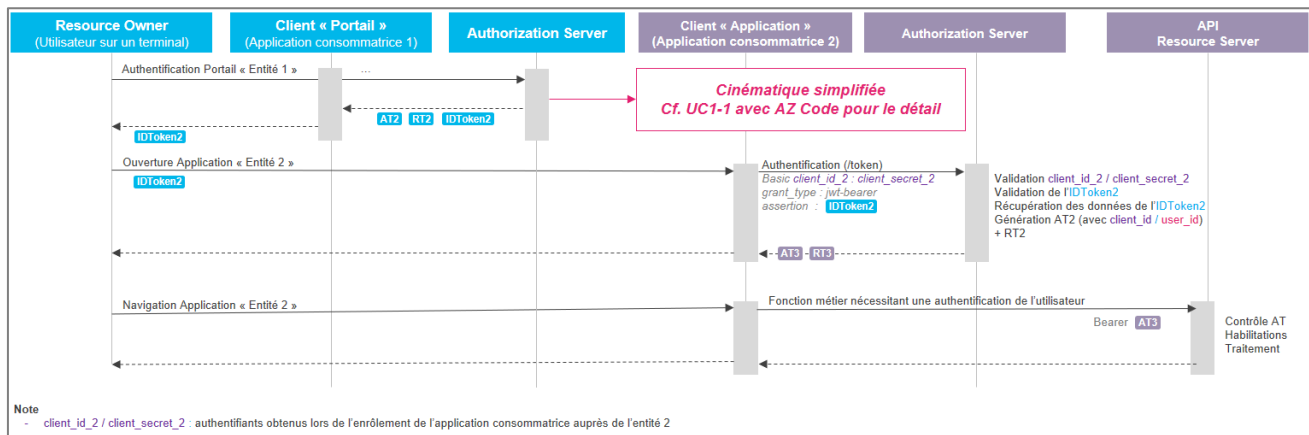
## 5.3.5.4.4.3 UC2-2 Authorization Code

Type d'usage / Exemple	
	<ul style="list-style-type: none"> <li>- Ce cas d'usage nécessite un enrôlement de l'application consommatrice auprès de l'entité 2.</li> <li>- L'utilisateur et la ressource étant sous la responsabilité de deux entités différentes, cette cinématique nécessite la transmission de la preuve d'authentification de l'utilisateur entre les 2 entités (jeton IDToken). La récupération de cet ID Token par l'entité cible pourra se faire suivant deux cinématiques détaillées ci-après</li> <li>- L'id token n'est délivré à l'application consommatrice que si elle est autorisée à demander le scope openid.</li> </ul>

### 5.3.5.4.4.3.1 UC2-2 AZ Code avec ID Token unique (Client et AZ Server)

Dans cette première possibilité de cinématique, l'ID Token reçu par l'application consommatrice (Client) est également transmis à l'Authorization Server de l'entité fournissant l'API. Cette ID Token produit par l'Entité 1 contiendra dans son attribut « audience » les deux applicatifs destinés à le recevoir (application Client et AZ Server).

Le schéma ci-dessous illustre cette 1<sup>ère</sup> cinématique d'authentification associée (sans NNI IHM) :



Si l'application consommatrice est accédée via NNI IHM ou autre SSO, cf. [5.5.2 Cas des applications compatibles NNI IHM](#)

#### Autres remarques :

- Le jeton IDToken n'est utilisé que pour initialiser la session d'authentification de l'utilisateur sur l'autorization server de l'Entité cible (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect). Ainsi, la durée de validité du jeton IDToken sera de quelques minutes.
- Après réception du jeton IDToken par l'application consommatrice, la génération des jetons d'accès à l'API est de la responsabilité de l'Entité hébergeant l'API et l'application. La

cinématique OAuth/OIDC proposée ici côté Entité cible n'est qu'une bonne pratique (non normatif).

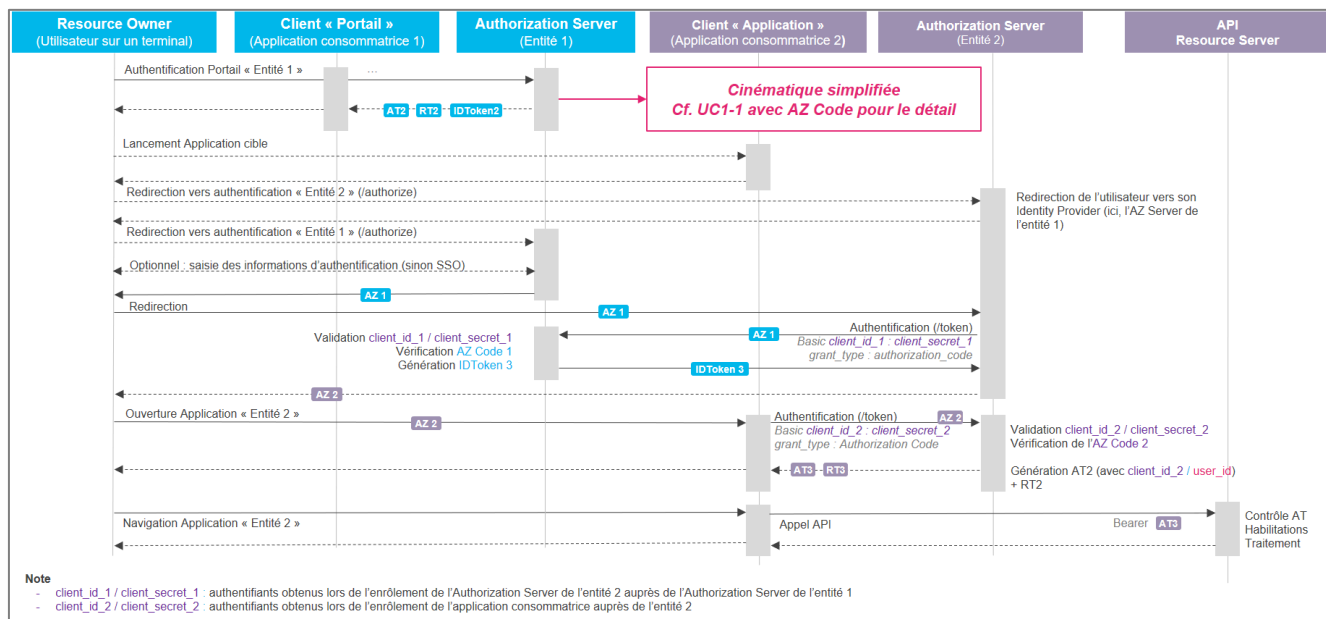
- Si l'Authorization Server violet de l'entité 2 est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

## 5.3.5.4.4.3.2 UC2-2 AZ Code avec deux ID Token (Client et AZ Server)

Dans cette deuxième possibilité de cinématique, l'application consommatrice (Client) reçoit un ID token qui lui est dédié et l'Authorization Server de l'entité fournissant l'API reçoit un 2<sup>ème</sup> ID Token dédié également. Chaque ID Token produit par l'Entité 1 contiendra dans son attribut « audience » le seul applicatif destiné à le recevoir (application Client ou AZ Server).

Le schéma ci-dessous illustre cette 2<sup>ème</sup> cinématique d'authentification associée à ce cas d'usage :



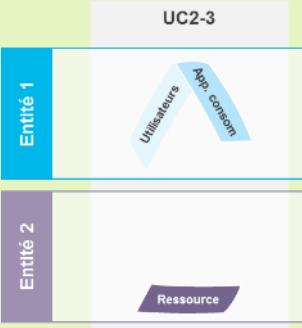
A noter que dans cette 2<sup>ème</sup> cinématique, l'Authorization Server (violet) de l'entité 2 joue le rôle de broker d'identité en redirigeant l'utilisateur vers son identity provider, à savoir l'Authorization Server (bleu) de l'entité 1.

Cette 2<sup>ème</sup> cinématique, bien que plus complexe, peut s'avérer plus simple à mettre en œuvre lorsque les deux Authorization Servers sont des composants progiciels (exemple : Keycloak) prenant automatique en charge les différents flux standards du protocole OpenID Connect.

Si l'application consommatrice est accédée via NNI IHM ou autre SSO, cf. [5.5.2 Cas des applications compatibles NNI IHM](#)

Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

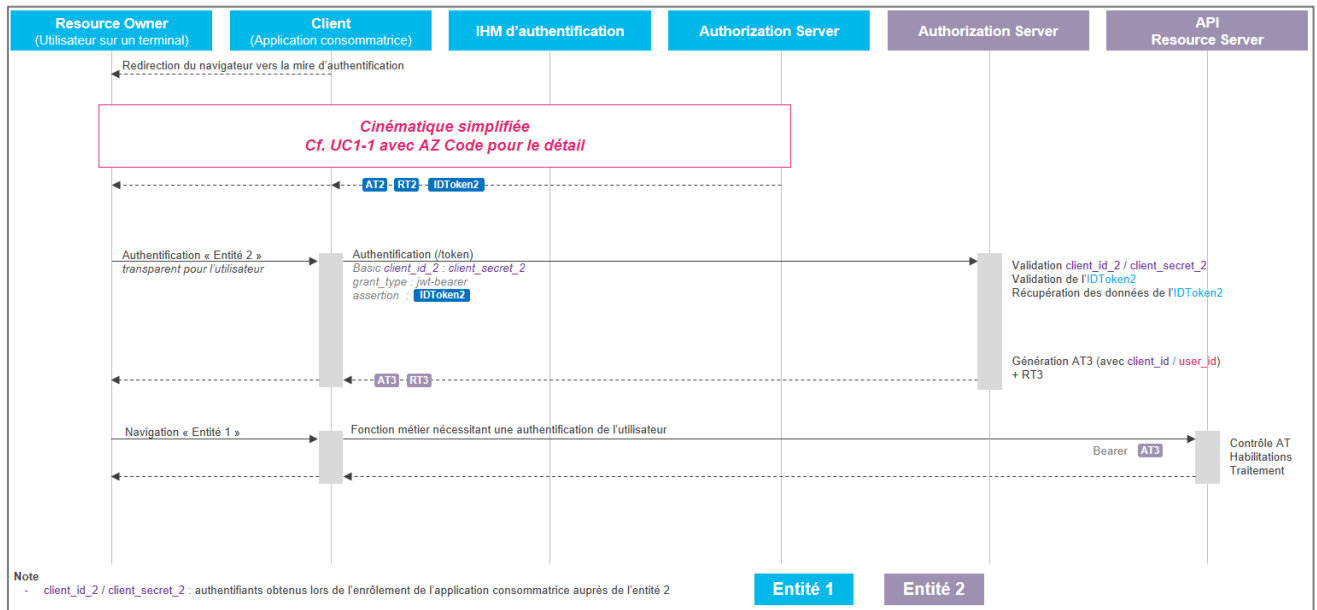
## 5.3.5.4.4.4 UC2-3 Authorization Code

Type d'usage / Exemple	
	<ul style="list-style-type: none"> <li>- Ce cas d'usage nécessite un enrôlement de l'application consommatrice auprès de l'entité 2.</li> <li>- L'utilisateur et la ressource étant sous la responsabilité de deux entités différentes, cette cinématique nécessite la transmission de la preuve d'authentification de l'utilisateur entre les 2 entités (jeton IDToken). La récupération de cet ID Token par l'entité cible pourra se faire suivant deux cinématiques détaillées ci-après.</li> <li>- L'id token n'est délivré à l'application consommatrice que si elle est autorisée à demander le scope openid.</li> <li>- Ce cas d'usage peut également s'appliquer lorsque l'Entité 1 est un <u>partenaire extérieur</u> souhaitant accéder à une API du Groupe (Entité 2) avec des utilisateurs s'identifiant chez le partenaire externe.</li> </ul>

### 5.3.5.4.4.4.1 UC2-3 AZ Code avec ID Token unique (Client et AZ Server)

Dans cette première possibilité de cinématique, l'ID Token reçu par l'application consommatrice (Client) est également transmis à l'Authorization Server de l'entité fournissant l'API. Cette ID Token produit par l'Entité 1 contiendra dans son attribut « audience » les deux applicatifs destinés à le recevoir (application Client et AZ Server).

Le schéma ci-dessous illustre cette 1<sup>ère</sup> cinématique d'authentification associée à ce cas d'usage :



## Autres remarques :

- Le jeton IDToken n'est utilisé que pour initialiser la session d'authentification de l'utilisateur sur l'autorization server de l'Entité cible (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect). Ainsi, la durée de validité du jeton IDToken sera de quelques minutes.
- Si l'Authorization Server violet de l'entité 2 est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'application Client. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

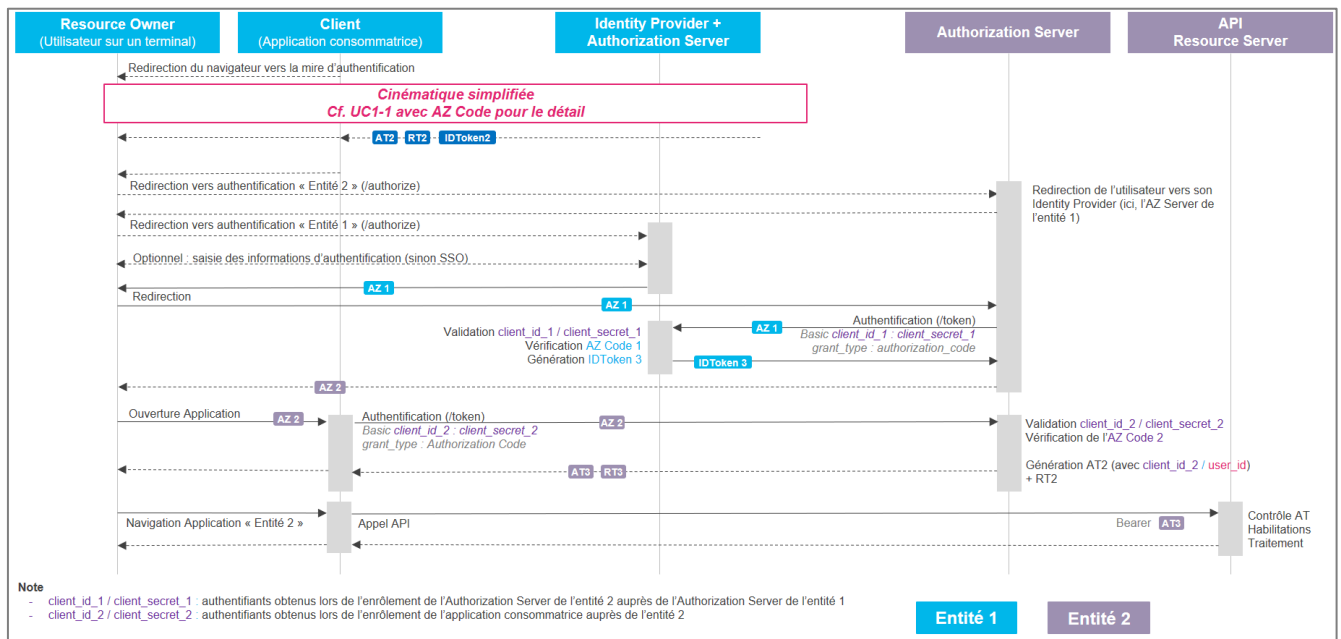
Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

### 5.3.5.4.4.2 UC2-3 AZ Code avec deux ID Token (Client et AZ Server)

Dans cette deuxième possibilité de cinématique, l'application consommatrice (Client) reçoit un ID token qui lui est dédié et l'Authorization Server de l'entité fournissant l'API reçoit un 2<sup>ème</sup> ID Token dédié également. Chaque ID Token produit par l'Entité 1 contiendra dans son attribut « audience » le seul applicatif destiné à le recevoir (application Client ou AZ Server).

Le schéma ci-dessous illustre cette 2<sup>ème</sup> cinématique d'authentification associée à ce cas d'usage :





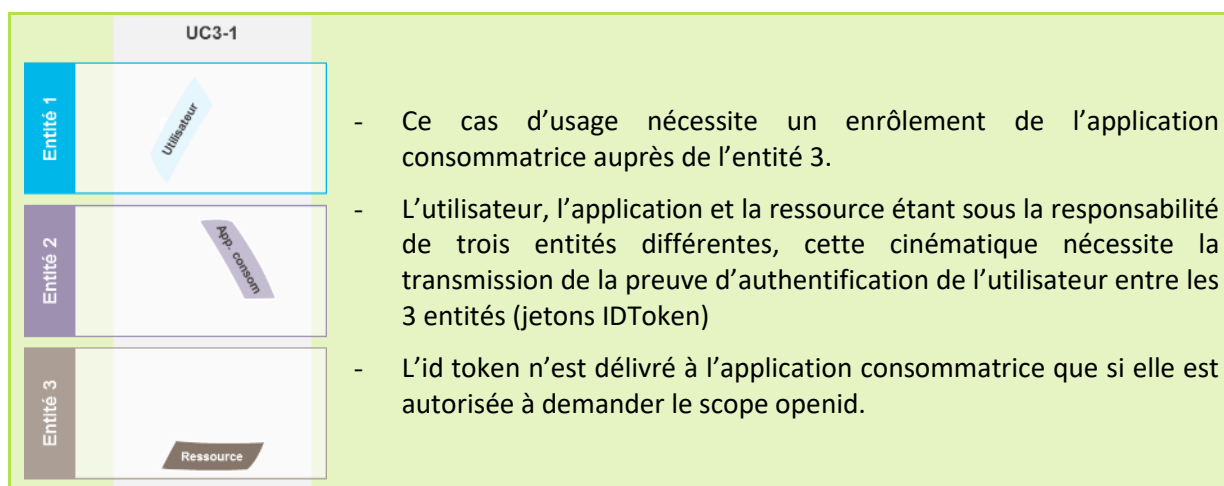
A noter que dans cette 2<sup>ème</sup> cinématique, l'Authorization Server (violet) de l'entité 2 joue le rôle de broker d'identité en redirigeant l'utilisateur vers son identity provider, à savoir l'Authorization Server (bleu) de l'entité 1.

Cette 2<sup>ème</sup> cinématique, bien que plus complexe, peut s'avérer plus simple à mettre en œuvre lorsque les deux Authorization Servers sont des composants progiciels (exemple : Keycloak) prenant automatique en charge les différents flux standards du protocole OpenID Connect.

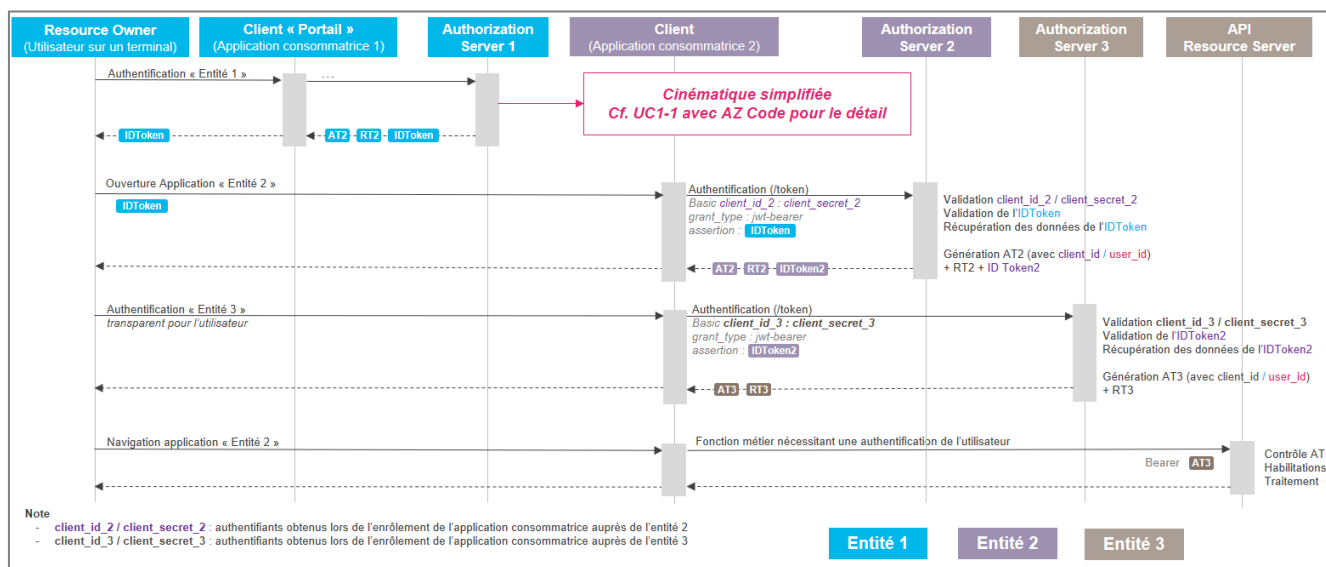
Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre 5.3.5.3.2. On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

## 5.3.5.4.4.5 UC3-1 Authorization Code

Type d'usage / Exemple



Le schéma ci-dessous illustre la cinématique d'authentification associée à ce cas d'usage (sans NNI IHM) :



Les cinématiques de renouvellement de jeton d'accès et de révocation de la session d'authentification sont identiques à celles décrites dans le chapitre [5.3.5.3.2](#). On notera néanmoins que la révocation de la session d'authentification vaut pour les deux types de jetons d'accès impliqués dans la cinématique.

Si l'application consommatrice est accédée via NNI IHM ou autre SSO, cf. [5.5.2 Cas des applications compatibles NNI IHM](#)

## Autres remarques :

- Les jetons IDToken ne sont utilisés que pour initialiser la session d'authentification de l'utilisateur sur les authorization server des Entités cibles (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect). Ainsi, la durée de validité des jetons IDToken sera de quelques minutes.
- Si les Authorization Server 2 ou 3 sont des Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-

ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».

### 5.3.5.4.5 Spécification de l'Access Token (Authorization Code)

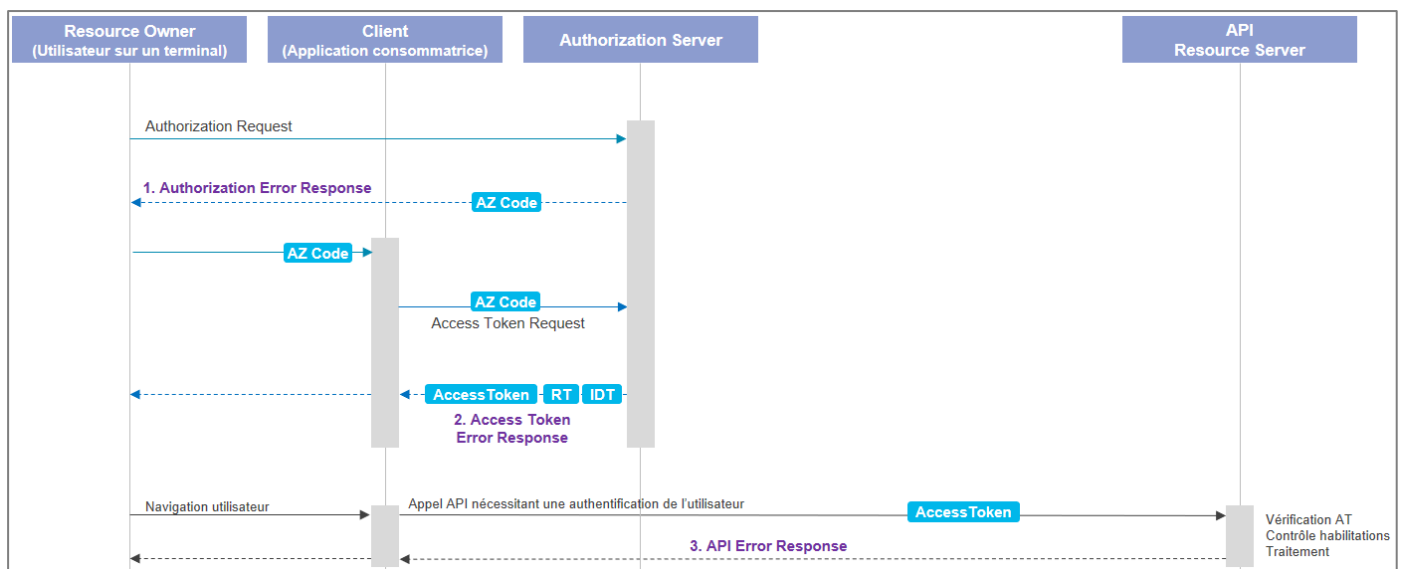
Se référer au chapitre 5.3.5.3.4

### 5.3.5.4.6 Spécification de l'ID Token (Authorization Code)

Se référer au chapitre 5.3.5.3.5

## 5.3.5.5 Cinématique générale et format des réponses pour les erreurs

### Cinématique générale (tout cas d'usage)



### Description détaillée des requêtes d'erreur :

#### 1. Authorization Error Response et Access Token Error Response :

Les 2 requêtes d'erreur ont le même format (Authorization Error Response, Access Token Error Response) :

```

HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{

```

```
"error": "CODE_ERREUR",  
"error_description": "DESCRIPTION_ERREUR", //optionnel  
"error_uri": "URI_ERREUR" //optionnel  
}
```

Conformément au standard OAuth2 ([RFC6749](#)), cette requête aura les caractéristiques suivantes :

- TLS (HTTPS) → Obligatoire
- Status Code HTTP → Obligatoire et entre 400 et 499
- Header « Cache-Control: no-store » → Obligatoire
- Header « Pragma: no-cache » → Obligatoire
- Paramètre « error » → Obligatoire et avec les valeurs définis dans la RFC6749
- Paramètre « error\_description » → Optionnel
- Paramètre « error\_uri » → Optionnel

Pour une description détaillée de ces paramètres : Cf. [5.3.5.8 Glossaire des données utilisées dans les réponses en erreur \(format JSON\)](#)

### 3. API Error Response :

Si l'application consommatrice a tenté d'accéder à l'API en ayant renseigné le header « Authorization », alors la réponse à une erreur d'authentification devra inclure un header « www-Authenticate » pour y indiquer le schéma d'authentification attendu et les éventuels code et message d'erreur.

- Si aucun Access Token n'est transmis, la réponse d'erreur est la suivante :

```
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Bearer realm="example"
```

- Si un Access Token est transmis, la réponse est la suivante pour les autres erreurs d'accès (access token non valide) :

```
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Bearer realm="A_RENSEIGNER",  
error="CODE_ERREUR",  
error_description="DESCRIPTION_ERREUR", //optionnel  
error_uri="URI_ERREUR" //optionnel
```

Conformément au standard OAuth2 Bearer Token Usage ([RFC6750](#)), cette requête aura les caractéristiques suivantes :

- TLS (HTTPS) → Obligatoire
- Status Code HTTP → Obligatoire et entre 400 et 499
- Paramètre « Authentication Scheme » → Obligatoire et « Bearer » si un Access Token est attendu
- Paramètre « error » → Obligatoire et avec les valeurs définies dans la RFC6750 et RFC6749
- Paramètre « error\_description » → Optionnel
- Paramètre « error\_uri » → Optionnel

Pour une description détaillée de ces paramètres : Cf. [5.3.5.8 Glossaire des données utilisées dans les réponses en erreur \(format JSON\)](#)

- Si un Access Token est transmis et qu'il est valide, mais que l'API souhaite retourner une erreur fonctionnelle (erreur HTTP 4xx) ou technique indépendante des contrôles d'accès (erreur HTTP 500, etc.), les informations sur l'erreur seront positionnées dans le BODY de la réponse. Pour une description détaillée, voir le §2.9 du document « Norme API – Bonnes pratiques de conception et utilisation de ressources » (RefAE\_NormeAPI\_BP\_Ressources.pdf) accessible sur cette [page](#).

## 5.3.5.6 Glossaire des données utilisées dans les jetons (format JSON)

### Header

Intitulé	Description
<b>alg</b>	<b>Algorithme de signature</b> Spécifie l'algorithme cryptographique utilisé pour la signature. Il doit être renseigné avec la valeur <b>RS256</b> (signature RSA avec une fonction de calcul de hash qui utilise l'algorithme SHA-256) pour un jeton JWS et <b>RSA-OAEP-256</b> pour un jeton JWE.
<b>typ</b>	<b>Media Type</b> Pour les Access Token, il aura pour valeur : « at+JWT »
<b>cty</b>	<b>Type de jeton</b> Indique le type de contenu du jeton. Il doit être renseigné avec la valeur "JWT" car il englobe un autre jeton JWT lié à la signature
<b>kid</b>	<b>DN Certificat x509 utilisé pour la signature</b> Kid (Key identifier) : correspond au DN du certificat ayant servi à signer. Donnée facultative, à utiliser uniquement pour les jetons JWT sans certificat.
<b>x5c</b>	<b>Certificat x509 utilisé pour la signature (généré par la PKI Groupe)</b> Contient le certificat qui renferme la clé publique correspondant à la clé privée utilisée pour signer le jeton.  <div style="border: 1px solid black; padding: 5px;"> <p>Lorsque cela est possible (au regard des contraintes de performance selon le volume échangé et en fonction des possibilités de traitement à la réception du jeton), il est préconisé de privilégier l'alimentation de l'attribut x5c ou x5u (ci-après) avec le certificat x509 pour avoir un jeton autoporteur et ainsi simplifier l'architecture et le processus de renouvellement du certificat.</p> <p>Lorsque l'on utilise cet attribut, afin d'éviter de transporter systématiquement la chaîne de certification complète dans le jeton, il est obligatoire d'installer le certificat racine de la PKI Groupe sur le composant en charge de la vérification du jeton.</p> </div>
<b>x5u</b>	Lien vers le certificat X509 de signature

	Contient l'url permettant d'accéder au certificat qui renferme la clé publique correspondant à la clé privée utilisée pour signer le jeton. Nécessite que l'Authorization Server expose une URI permettant de récupérer le certificat de signature.
<b>x5t#S256</b>	<p><b>Empreinte SHA-256 certificat x509 de signature</b></p> <p>Contient l'empreinte du certificat de signature utilisé pour signer le jeton. Cette empreinte est une empreinte SHA-256 encodée au format « base64url ».</p> <p>A noter, que cet attribut « x5t#S256 » a remplacé l'attribut « x5t » qui utilisait une empreinte SHA-1, aujourd'hui dépréciée.</p>

## Body

Intitulé	Description
<b>iss</b>	<p><b>Émetteur du jeton</b></p> <p>L'adresse du serveur qui a généré les informations de sécurité. Elle est renseignée de la manière suivante : "https://&lt;hostname_du_serveur_ayant_émis_le_jeton&gt;[:port/&lt;path_du_service_de_création_des_jetons&gt;]" (le hostname est obligatoire, le port et le path sont facultatifs).</p> <p>Il n'est pas obligatoire d'exposer l'url pour qu'elle puisse être vérifiée.</p>
<b>sub</b>	<p><b>Identifiant du porteur</b></p> <p>L'identifiant du bénéficiaire des données de sécurité. Il est renseigné en calculant le HASH de la concaténation des trois attributs « userType », « userUomCode » et « userId » (dans cet ordre). L'algorithme à utiliser pour calculer le condensé (hash) est à minima SHA2-256. Pour un client multi bancarisé, l'uom de rattachement à retenir est l'entité pour laquelle le client s'est authentifié.</p>
<b>aud</b>	<p><b>Audience(s) du jeton</b></p> <p>Il s'agit de l'ensemble des applicatifs destinataires (API ou ressources ou applications) du jeton de sécurité. L'audience contient une liste de valeurs de type StringOrURI, chaque valeur constituant l'identifiant unique d'une API ou d'une ressource ou, de manière plus générale, d'un composant applicatif.</p> <p>L'audience sera obligatoirement renseignée dans le contenu du jeton d'accès produit et, obligatoirement contrôlé, à la réception par l'API ou le composant applicatif (vérifier que l'identifiant de l'applicatif accédé apparaît bien dans la liste des valeurs de l'audience).</p> <p>A noter que dans les jetons de type IDToken la valeur de l'audience inclura le Client_id de l'application cliente (conformément au standard OpenID connect).</p> <p>Format de la valeur « aud » tel que spécifié dans la RFC7519 :</p> <ul style="list-style-type: none"> <li>- Une « String ou URI » si l'audience ne contient qu'un seul identifiant</li> <li>- Un tableau de « String ou URI » si l'audience ne contient une liste de valeur <ul style="list-style-type: none"> <li>o Dans les jetons en JSON : ["id1", "id2", "id3"]</li> </ul> </li> </ul> <p>Cf. <a href="#">Glossaire des données utilisées dans les jetons</a></p>
<b>exp</b>	<p><b>Heure d'expiration du jeton</b></p> <p>Le format est NumericDate, qui exprime le nombre de secondes écoulées depuis le 01/01/1970 en temps universel (1970-01-01T00:00:00Z UTC).</p>
<b>client_id</b>	<p><b>Identifiant de l'application consommatrice</b></p> <p>Identifiant obtenu par l'application consommatrice lors de son enrôlement auprès du fournisseur de l'API (via un portail d'enrôlement).</p> <p>Cet identifiant a pour format une chaîne de caractères dont les caractéristiques sont fonction de la solution d'enrôlement utilisée (exemple de format : SHA-256 d'une chaîne de caractères composée d'éléments garantissant l'unicité).</p>
<b>iat</b>	<p><b>Heure de délivrance du jeton</b></p> <p>Le format est NumericDate, qui exprime le nombre de secondes écoulées depuis le 01/01/1970 en temps universel (1970-01-01T00:00:00Z UTC).</p>
<b>auth_time</b>	<p><b>Heure de l'authentification</b></p> <p>La date et l'heure de l'authentification de l'utilisateur de l'application qui appelle le service. Le format est NumericDate, qui exprime le nombre de secondes écoulées depuis le 01/01/1970 en temps universel (1970-01-01T00:00:00Z UTC). Pour un échange entre serveurs, sans utilisateur, la valeur à utiliser est "none".</p>
<b>jti</b>	<p><b>Identifiant unique de jeton de sécurité</b></p> <p>Chaîne de caractères contenant une valeur pseudo-aléatoire dans un intervalle de valeurs. L'utilisation d'un « jti » et son contrôle par le fournisseur de ressources permettent de s'assurer que la requête ne pourra être utilisée qu'une seule fois sur une ressource donnée durant la plage de validité des informations de sécurité. Le contrôle est à réaliser par le destinataire de la requête : Il doit stocker les identifiants « jti » reçus pendant la période de validité de la requête (période de validité des informations de sécurité) et vérifier qu'il n'y a pas doublon.</p> <p>L'alimentation du jti par l'émetteur du jeton est obligatoire (facultatif pour l'ID Token). Son utilisation par le fournisseur de ressource est facultative. Cela peut s'avérer opportun dans les cas où le jeton est susceptible</p>

	<p>d'être utilisé pour des transactions présentant un risque de rejeu : création d'instance, virement, transfert de fonds....</p> <p><b>Un identifiant « jti » doit être généré et sa non réutilisation doit être vérifiée lorsque les exigences de confidentialité et d'intégrité sont élevées (<math>I \geq 3</math> et <math>C \geq 3</math>).</b></p> <p>Lorsqu'il est utilisé, il contient un identifiant unique universel (UUID) de version 4 : un identifiant construit à partir de nombres aléatoire. C'est un nombre de 16 octets représenté sous la forme de 32 digits, regroupés en 5 groupes séparés par des tirets. La longueur totale de la chaîne est donc de 36 caractères.</p>
nonce	<p><b>Identifiant non prédictible transmis par l'application Client lors d'une demande d'ID Token.</b></p> <p>Attribut uniquement présent dans le jeton ID Token.</p> <p>Sa mise en œuvre est décrite dans la spécification OpenID Connect (<a href="https://openid.net/specs/openid-connect-core-1_0.html">https://openid.net/specs/openid-connect-core-1_0.html</a>).</p> <p>Il s'agit d'un paramètre de sécurité permettant à l'application consommatrice (Client) de vérifier que l'ID Token reçu correspond bien à l'« authorization request » initiale. Dans nos cas d'usage, l'utilisation du nonce est optionnelle. Il offre un niveau de sécurité supplémentaire.</p> <p>Pour information (extrait de la spécification OpenID Connect), la cinématique à mettre en œuvre pour utiliser ce nonce OIDC :</p> <ol style="list-style-type: none"> <li>1. Application consommatrice (Client) génère une valeur « nonce » aléatoire et la conserve en mémoire</li> <li>2. Elle ajoute ce paramètre « nonce » dans la requête d'autorisation envoyée vers l'Authorization Server, si scope "openid" demandé</li> <li>3. Une fois l'authentification utilisateur effectuée, l'Authorization Server conserve le lien entre l'application consommatrice (Client_id) et le nonce et redirige l'utilisateur vers l'application consommatrice avec l'AZ Code</li> <li>4. L'application consommatrice demande les tokens (access token request) et reçoit un ID token contenant le nonce initialement envoyé</li> <li>5. L'application consommatrice contrôle que le nonce de l'ID Token est bien égale au nonce gardé en mémoire.</li> </ol>
acr	<p><b>Niveau de confiance de la preuve d'authentification</b></p> <p>La nomenclature des valeurs est définie dans le paragraphe 5.3.6.</p> <p>Pour un échange entre serveurs, sans utilisateur, la valeur à utiliser est 10.</p>
amr	<p><b>Méthode d'authentification</b></p> <p>La nomenclature des valeurs est définie par la RFC 8176 (<a href="https://tools.ietf.org/html/rfc8176">https://tools.ietf.org/html/rfc8176</a>)</p> <p>Pour un échange entre serveurs, sans utilisateur humain, la valeur à utiliser est une liste à choix multiple parmi « pwd » (si utilisation du client_secret), « swk » ou « hwk » (si utilisation d'un certificat software ou hardware).</p>
scope	<p><b>Périmètre des droits (habilitations) délégués à l'application consommatrice, pour accéder à l'API</b></p> <p>Le rôle fonctionnel de ce paramètre est décrit au <a href="#">§5.3.5.1.3 Cas des profils d'habilitation applicatifs inter-entités : scope et user profile</a></p> <p>Le paramètre « scope » a pour valeur une liste de chaîne de caractères délimitées par un espace (<a href="https://tools.ietf.org/html/rfc6749#section-3.3">https://tools.ietf.org/html/rfc6749#section-3.3</a>).</p> <p>Par exemple : "scope1 scope2 scope3"</p> <p>Dans le cadre de la mise en œuvre d'OpenID Connect, certaines valeurs de scope sont définies dans le standard : « openid », « profile », « email », « phone », etc. (<a href="https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims">https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims</a>).</p> <p>Dans le cadre des échanges Groupe, la valeur de scope « user-id » et « user-ca » ont été définis (Cf. <a href="#">5.3.5.3.5 Spécification de l'id token (ROPC)</a>).</p> <p>D'autres valeurs de scope peuvent être définies par les entités.</p> <p>Seules les « scope » autorisés pour l'application consommatrice seront ajoutés dans le jeton Access Token produit.</p>
authorization_context	<b>Contexte d'autorisation</b>
session_id	<b>Identifiant de la session d'authentification initiale de l'utilisateur final</b>
user_type	<p><b>Type d'utilisateur</b></p> <p>Le type d'utilisateur (d'acteur) à l'initiative de l'appel. Donnée obligatoire. Chaîne de caractères de longueur 2 caractères alphanumériques.</p> <ul style="list-style-type: none"> <li>- 01 : Agent</li> <li>- 02 : Personne physique connue du Système d'Information</li> <li>- 03 : Personne physique non connue du Système d'Information (intervenant au travers d'un contrat d'entreprise)</li> <li>- 04 : Prospect</li> <li>- 05 : Traitement informatique non déclenché par un humain (Automate)</li> </ul>

<b>user_uom_code</b>	<p><b>UO Métier d'appartenance de l'utilisateur</b></p> <p>Chaîne de 5 caractères alphanumériques cadrés à gauche, renseignée selon la nomenclature des <a href="#">UO Métier CAM0303</a>.</p> <p>Pour les applications "déclenchées par un humain" (par opposition aux traitements batch), cette donnée doit être renseignée en fonction du type d'utilisateur trouvé selon le tableau ci-dessous :</p> <ul style="list-style-type: none"> <li>- 01, 02, 03 : Identifiant national de l'entité gestionnaire de l'acteur</li> <li>- 04 : Le numéro de l'entité qui héberge l'application à l'initiative de l'appel.</li> <li>- 05 : Pour les traitements non déclenchés par un humain, elle n'est pas renseignée.</li> </ul>
<b>user_id</b>	<p><b>Identifiant de l'utilisateur</b></p> <p>Identifiant correspondant à l'acteur authentifié pour lequel l'échange est réalisé.</p> <p>Un acteur représente une personne physique ou un automate, qui réalise des tâches dans le cadre de processus métiers.</p> <p>Un collaborateur est un acteur de type personne physique, employé par une UO Métier du Groupe, renseigné dans l'attribut userUomCode.</p> <p>Un collaborateur peut travailler pour le compte de l'UO Métier qui l'emploie et/ou d'autres UO Métier (exemple : cas des coopérations entre Caisses Régionales).</p> <p>Doit être renseigné en fonction du type d'utilisateur selon le tableau de correspondance ci-dessous :</p> <ul style="list-style-type: none"> <li>- 01 : identifiant de l'agent</li> <li>- 02 : identifiant de l'utilisateur personne physique</li> <li>- 03 : identifiant de l'élément de contrat utilisé pour l'identification</li> <li>- 04 : Sans Objet. Peut être remplacé par des données diverses, éventuellement une saisie applicative.</li> <li>- 05 : Le nom du job</li> </ul>
<b>user_entitlements</b>	<p><b>Accréditations de l'utilisateur</b></p> <p>Sous-structure regroupant les caractéristiques d'habilitation fonctionnelle de l'acteur (tel que définie au § 5.1) pour une uom. Ces données permettent, au travers du profil d'habilitation et des compléments, de définir les fonctions que l'acteur peut utiliser (i.e. les autorisations) et les conditions d'utilisations de ces fonctions, en termes de périmètre de données et de contraintes dans l'utilisation (voir <a href="#">IDHAB</a> pour plus de détails).</p> <p>L'utilisateur peut avoir une habilitation pour plusieurs uom. Il aura un élément Accreditations pour chacune. Cette structure est obligatoire pour les cas d'usages d'échanges inter-entités impliquant un utilisateur de type collaborateur ou automate.</p> <p>Lorsqu'elle est présente, elle est référentielle et fait autorité. Lorsqu'elle n'est pas renseignée ou renseignée avec une valeur à null, cela signifie qu'elle n'a pas été définie pour le serveur d'autorisations considéré. Le cas échéant, les valeurs sont à négocier entre l'application consommatrice et les ressources.</p>
<b>uom_code</b>	<p><b>Code de l'UO Métier (identifiant nationale de l'Entité) pour laquelle l'utilisateur travaille.</b></p> <p><b>Uom_code</b> doit refléter à chaque instant l'uom sur laquelle l'utilisateur travaille. En cas de changement d'uom pendant la session de l'utilisateur, il revient à l'application de faire varier la valeur de l'<b>uom_code</b> pour refléter ce changement.</p> <p>Chaîne de 5 caractères alphanumériques cadrés à gauche, renseignée par l'application selon la nomenclature des <a href="#">UO Métier CAM0303</a>.</p>
<b>user_profile</b>	<p><b>Profil d'habilitation fonctionnelle de l'utilisateur pour l'uom considérée et l'échange réalisé.</b></p> <p>Un profil permet d'utiliser des fonctions au sein d'un espace d'habilitation (un ensemble d'applications) ; il factorise les permissions attribuées à un ou plusieurs rôles au sein de cet espace (voir <a href="#">IDHAB</a> pour plus de détails). Chaque profil est unique au sein d'un espace d'habilitation. Au niveau applicatif, un profil permet :</p> <ul style="list-style-type: none"> <li>- D'utiliser des opérations de services</li> <li>- De composer une IHM en cohérence avec les permissions qu'il accorde et de vérifier que chaque action réalisée par l'acteur est conforme à celles-ci</li> </ul> <p>Le rôle fonctionnel de ce paramètre par rapport au paramètre « scope » est décrit au <a href="#">§5.3.5.1.3 Cas des profils d'habilitation applicatifs inter-entités : scope et user_profile</a></p> <p><b>Remarque :</b> les profils doivent être conformes aux exigences exprimées dans les livrables d'échanges. L'application appelée doit vérifier que le profil la concerne bien pour éviter les utilisations abusives (rejeu de message avec usurpation).</p>
<b>additional_element</b>	<p><b>Données complémentaires d'habilitation</b></p> <p>Sous-structure permettant de matérialiser les restrictions ou extensions d'habilitations (ou habilitations fines) concernant l'utilisateur sur les données manipulées par les applications.</p> <p>Elle comprend une clé indiquant le nom de la donnée sur laquelle porte le complément d'habilitation, et une liste de valeurs indiquant le complément en tant que tel. La liste peut se limiter à une valeur.</p> <p>L'utilisateur peut avoir, sur une même application ou sur plusieurs, plusieurs occurrences « additional_element » de définis.</p>
<b>name</b>	Nom caractérisant la nature de l'élément complémentaire affinant le périmètre



value

Valeur ou liste des valeurs associée(s) à la variable indiquée ci-dessus.

## 5.3.5.7 Glossaire des données HTTP des échanges OAuth/OIDC

Intitulé	Description
<b>grant_type</b>	<p><b>Cinématique OAuth</b></p> <p>La cinématique utilisée doit être indiquée dans un paramètre « grant_type » de la requête de demande d'un jeton d'accès (access token request ou refresh token request) initiée par l'application Client et à destination de l'autorization server.</p> <p>Les valeurs possibles, dans le cadre de la norme API, sont :</p> <ul style="list-style-type: none"> <li>- « client_credentials »</li> <li>- « authorization_code »</li> <li>- « password » pour ROPC</li> <li>- « refresh_token » pour demander le renouvellement du jeton access_token</li> <li>- « urn:ietf:params:oauth:grant-type:jwt-bearer » pour demander un jeton OAuth à partir d'un jeton JWT comme preuve de l'authentification de l'utilisateur (grant_type à utiliser lorsque les jeton JWT IDToken est transmis). Cf. <a href="#">RFC7523</a></li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request, Refresh Token Request</p>
<b>client_id</b>	<p><b>Identifiant de l'application consommatrice</b></p> <p>Identifiant obtenu par l'application consommatrice lors de son enrôlement auprès du fournisseur de l'API (via un portail d'enrôlement).</p> <p>Cet identifiant a pour format une chaîne de caractères dont les caractéristiques sont fonction de la solution d'enrôlement utilisée (exemple de format : SHA-256 d'une chaîne de caractères composée d'éléments garantissant l'unicité).</p> <p><b>Paramètre utilisé dans les requêtes :</b> Authorization Request, Access Token Request (si HTTP Basic Auth non utilisé), Refresh Token Request (si HTTP Basic Auth non utilisé), Revoke Token Request (si HTTP Basic Auth non utilisé)</p>
<b>client_assertion</b>	<p><b>Jeton d'authentification de l'application consommatrice</b></p> <p>Il est utilisé pour authentifier l'application consommatrice (Client) auprès de l'Authorization Server.</p> <p>Il est au format JSON Web Token. La signature du JWT pourra s'effectuer soit grâce à une Clé privée / Clé publique (préconisé), soit grâce au « HMAC SHA algorithm » en s'appuyant sur le client_secret pour générer la HMAC.</p> <p>Les données contenues dans le jeton JWT sont décrites au <a href="#">§5.3.4.2</a>.</p> <p>Voir <a href="#">RFC7523</a> et <a href="#">OpenID Connect Core 1.0</a>, pour les spécifications détaillées</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request (si HTTP Basic Auth non utilisé), Refresh Token Request (si HTTP Basic Auth non utilisé), Revoke Token Request (si HTTP Basic Auth non utilisé)</p>
<b>client_assertion_type</b>	<p><b>Type de Jeton d'authentification de l'application consommatrice</b></p> <p>Il est utilisé en complément du paramètre « client_assertion » pour indiquer son type.</p> <p>Dans nos cas d'usage, le « client_assertion » sera toujours au format JWT, donc, la valeur de client_assertion_type sera toujours égale à :</p> <p>"urn:ietf:params:oauth:client-assertion-type:jwt-bearer"</p> <p>Voir <a href="#">RFC7523</a> et <a href="#">OpenID Connect Core 1.0</a>, pour les spécifications détaillées</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request (si HTTP Basic Auth non utilisé), Refresh Token Request (si HTTP Basic Auth non utilisé), Revoke Token Request (si HTTP Basic Auth non utilisé)</p>
<b>Header "Authorization : Basic"</b>	<p><b>HTTP Basic Access Authentication</b></p> <p>L'en-tête HTTP « Authorization » est utilisée pour transmettre le client_id et le client_secret à l'Authorization Server. Celui-ci doit contenir la méthode utilisée (Basic) suivie de la représentation en Base64 du nom de l'utilisateur et du mot de passe séparés par le caractère « : » (deux-points)</p> <p><u>Exemple :</u></p>

	<p>Authorization: Basic QWxhZGRpbjpvGVuIHNIc2FtZQ==</p> <p>L'utilisation de l'HTTP Basic Auth est recommandée pour transmettre les Client_id et Client_secret de l'application consommatrice. Une autre solution possible consiste à transmettre les paramètres « client_id » et « client_secret » dans le corps de la requête.</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request, Revoke Token Request, Revoke Token Request</p>
Header "Authorization : Bearer"	<p><b>HTTP Authentification via un jeton d'accès (access token)</b></p> <p>L'en-tête HTTP « Authorization » est utilisée pour transmettre l'access token aux API.</p> <p><u>Exemple :</u></p> <p>Authorization: Bearer AbCdEf123456AbCdEf123456</p> <p>L'utilisation du header HTTP Authorization est recommandée pour transmettre l'Access Token à l'API. A noter, que sous certaines conditions (RFC6750), il est également possible de transmettre l'Access Token dans le corps de la requête (via un paramètre « access_token »).</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> API Request</p>
scope	<p><b>Périmètre des droits (habilitations) délégués à l'application consommatrice, pour accéder à l'API</b></p> <p>Ce paramètre HTTP permet à l'application consommatrice d'effectuer une demande de « scope » pour sa requête vers l'Authorization Server. Les « scope » non autorisés pour cette application seront ignorés et les autres seront ajoutés dans l'attribut « scope » du jeton produit.</p> <p>Le rôle fonctionnel de ce paramètre est décrit au <a href="#">§5.3.5.1.3 Cas des profils d'habilitation applicatifs inter-entités : scope et user profile</a></p> <p>Le paramètre « scope » a pour valeur une liste de chaîne de caractères délimitées par un espace (<a href="https://tools.ietf.org/html/rfc6749#section-3.3">https://tools.ietf.org/html/rfc6749#section-3.3</a>). Par exemple : "scope1 scope2 scope3"</p> <p>Dans le cadre de la mise en œuvre d'OpenID Connect, certaines valeurs de scope sont définies dans le standard : « openid », « profile », « email », « phone », etc. (<a href="https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims">https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims</a>). Dans le cadre des échanges Groupe, la valeur de scope « user-id » et « user-ca » ont été définies (Cf. <a href="#">5.3.5.3.5 Spécification de l'id token (ROPC)</a>). D'autres valeurs de scope peuvent être définies par les entités.</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Request, Authorization Response, Access Token Request, Access Token Response</p>
redirect_uri	<p><b>URL de l'application consommatrice (i.e. application Cliente)</b></p> <p>Cet URL est utilisé dans la cinématique « Authorization Code Grant » pour permettre à l'Authorization Server de rediriger l'utilisateur vers l'application consommatrice après l'avoir identifié et authentifié (et, éventuellement, recueilli son consentement).</p> <p>Ce paramètre est au format chaîne de caractère et contient une URL absolu (http://...)</p> <p>Lorsque ce paramètre est transmis par une application consommatrice à l'Authorization Server, il est recommandé que celui-ci vérifie que le nom de domaine associé à la "redirect_uri" appartienne bien à l'application consommatrice.</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Request, Access Token Request</p>
response_type	<p><b>Type de réponse demandée</b></p> <p>Ce paramètre est utilisé dans la cinématique « Authorization Code Grant » pour solliciter un Authorization Code auprès de l'Authorization Server via la requête d'autorisation.</p> <p>Sa valeur sera toujours égale à « code »</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Request</p>
code	<p><b>Authorization Code</b></p>

	<p>Ce paramètre est utilisé dans la cinématique « Authorization Code Grant » pour transmettre l'Authorization Code, via HTTP, auprès de l'Authorization Server via la requête de demande d'un Access Token.</p> <p>Ce code doit avoir un délai d'expiration court pour limiter les risques de vol. Il est recommandé de lui attribuer une durée de vie maximum de 10 minutes.</p> <p><b>Cf. RFC correspondante :</b> <a href="https://tools.ietf.org/html/rfc6749">https://tools.ietf.org/html/rfc6749</a></p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Response, Access Token Request</p>
state	<p><b>Paramètre de sécurité permettant à l'application consommatrice de conserver un état pour relier la requête d'autorisation avec la réponse reçue.</b></p> <p>Comme décrit dans la RFC6749 (<a href="https://tools.ietf.org/html/rfc6749">https://tools.ietf.org/html/rfc6749</a>) et en application de la RS19 de l'analyse de risque MESARI « Sécurité des appels d'API », la mise en œuvre de cette mesure de sécurité, de ce paramètre, est préconisée.</p> <p>Le format du paramètre « state » est une chaîne de caractère aléatoire non prédictible de taille suffisante pour limiter raisonnablement les risques de collision (en générale une dizaine de caractères).</p> <p>Etape à mettre en œuvre par l'application consommatrice (cliente) :</p> <ul style="list-style-type: none"> <li>- Générer une valeur « state » aléatoire et la conserver en mémoire (côté serveur)</li> <li>- Ajouter ce paramètre « state » dans la requête d'autorisation envoyée vers l'Authorization Server</li> <li>- Une fois l'authentification utilisateur effectuée, l'Authorization Server doit rediriger l'utilisateur vers l'application consommatrice en incluant ce même paramètre « state » dans la requête sans le modifier</li> <li>- Une fois l'utilisateur redirigé à nouveau vers elle, l'application consommatrice doit vérifier que le paramètre « state » a bien la même valeur que celui envoyé.</li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Request, Authorization Response</p>
nonce	<p><b>Identifiant non prédictible transmis par l'application Client lors d'une demande d'ID Token.</b></p> <p>Sa mise en œuvre est décrite dans la spécification OpenID Connect (<a href="https://openid.net/specs/openid-connect-core-1_0.html">https://openid.net/specs/openid-connect-core-1_0.html</a>).</p> <p>Il s'agit d'un paramètre de sécurité permettant à l'application consommatrice (Client) de vérifier que l'ID Token reçu correspond bien à l'« authorization request » initiale. Dans nos cas d'usage, l'utilisation du nonce est optionnelle. Il offre un niveau de sécurité supplémentaire.</p> <p>Pour information (extrait de la spécification OpenID Connect), la cinématique à mettre en œuvre pour utiliser ce nonce OIDC :</p> <ol style="list-style-type: none"> <li>1. Application consommatrice (Client) génère une valeur « nonce » aléatoire et la conserve en mémoire</li> <li>2. Elle ajoute ce paramètre « nonce » dans la requête d'autorisation envoyée vers l'Authorization Server, si scope "openid" demandé</li> <li>3. Une fois l'authentification utilisateur effectuée, l'Authorization Server conserve le lien entre l'application consommatrice (Client_id) et le nonce et redirige l'utilisateur vers l'application consommatrice avec l'AZ Code</li> <li>4. L'application consommatrice demande les tokens (access token request) et reçoit un ID token contenant le nonce initialement envoyé</li> <li>5. L'application consommatrice contrôle que le nonce de l'ID Token est bien égale au nonce gardé en mémoire.</li> </ol>
access_token	<p><b>Jeton d'accès aux API</b></p> <p>Jeton produit par l'Authorization Server et déléguant les autorisations d'accès aux API.</p> <p><a href="#">Cf. 5.3.3 Types de Jetons</a></p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Response, Access Token Response</p>
refresh_token	<p><b>Jeton de renouvellement</b></p> <p>Ce jeton est utilisé pour renouveler le jeton d'accès aux API (Access Token)</p>

<p><a href="#">Cf. 5.3.3 Types de Jetons</a></p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request, Access Token Response</p>	
<b>token_type</b>	<p><b>Type de jeton</b> Type de jeton retourné. Dans la norme API, la valeur de ce paramètre sera toujours égale à « bearer ».</p> <p><b>Cf. RFC correspondante :</b> <a href="https://tools.ietf.org/html/rfc6750">https://tools.ietf.org/html/rfc6750</a></p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Response, Access Token Response</p>
<b>expires_in</b>	<p><b>Durée de vie (en secondes) de l'Access Token</b> Ce paramètre HTTP précise la durée de vie du jeton pour éviter d'avoir à analyser le jeton pour la connaître. Cependant, la valeur faisant foi, d'un point de vue sécurité, est la durée de vie contenue dans le jeton (car signée). Ainsi, le paramètre « expires_in » doit être utilisé que pour des optimisations ergonomiques (permettre au navigateur d'éviter d'envoyer un access_token expiré).</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Authorization Response, Access Token Response</p>
<b>username</b>	<p><b>Identifiant de l'utilisateur</b> Paramètre permettant de transférer le login de l'utilisateur dans le cadre d'une cinématique ROPC</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request</p>
<b>password</b>	<p><b>Mot de passe de l'utilisateur</b> Paramètre permettant de transférer le mot de passe de l'utilisateur dans le cadre d'une cinématique ROPC</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request</p>
<b>assertion</b>	<p><b>Preuve d'authentification de l'utilisateur (Jeton JWT)</b> Paramètre permettant de transférer la preuve d'authentification (SSO) de l'utilisateur sous la forme d'un jeton au format JWT (IDToken) dans le cadre d'une demande d'Access Token</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Access Token Request with IDToken</p>
<b>token</b>	<p><b>Jeton à révoquer</b> Paramètre http utilisé pour transmettre le jeton à révoquer lors d'une « Revoke Token Request ».</p> <p>A noter que :</p> <ul style="list-style-type: none"> <li>- Si le jeton à révoquer est un refresh_token, alors les access_token associés devraient également être révoqués.</li> <li>- Si le jeton à révoquer est un access_token, alors le refresh_token associé peut être révoqué également.</li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Revoke Token Request</p>
<b>token_type_hint</b>	<p><b>Type de jeton à révoquer</b> Ce paramètre optionnel permet d'indiquer le type du jeton à révoquer lors d'une « Revoke Token Request ».</p> <p>Les valeurs possibles sont : "refresh_token", "access_token"</p> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> Revoke Token Request</p>

### 5.3.5.8 Glossaire des données utilisées dans les réponses en erreur (format JSON)

Le serveur répond en utilisant le code erreur HTTP approprié (en général : 400, 401, 403, 405, etc.), puis alimente, dans le body, une structure JSON décrivant l'erreur avec les variables décrites ci-dessous.

Intitulé	Description
<b>error</b>	<p><b>Code d'erreur</b> Paramètre obligatoirement présent lors d'une réponse en erreur. Il indique le code de l'erreur. La RFC OAuth2 précise la liste des valeurs possibles.</p> <p><b>Les valeurs d'erreurs possibles sont décrites dans les RFC :</b> « invalid_request », « invalid_client », « invalid_grant », « unauthorized_client », « unsupported_grant_type », « invalid_token », « insufficient_scope », « access_denied », « unsupported_response_type », « server_error », « temporarily_unavailable », etc.</p> <p><b>Cf. RFC correspondantes :</b></p> <ul style="list-style-type: none"> <li>- <a href="https://tools.ietf.org/html/rfc6749#page-45">https://tools.ietf.org/html/rfc6749#page-45</a></li> <li>- <a href="https://tools.ietf.org/html/rfc6750#section-3.1">https://tools.ietf.org/html/rfc6750#section-3.1</a></li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> API Response, Authorization Response, Access Token Response</p>
<b>error_description</b>	<p><b>Description de l'erreur</b> Paramètre optionnel présent lors d'une réponse en erreur. Il indique la description de l'erreur. Cet attribut a pour objectif de fournir aux développeurs une description suffisamment explicite, mais n'est pas destinée à être affichée aux utilisateurs finaux</p> <p><b>Cf. RFC correspondantes :</b></p> <ul style="list-style-type: none"> <li>- <a href="https://tools.ietf.org/html/rfc6749#page-45">https://tools.ietf.org/html/rfc6749#page-45</a></li> <li>- <a href="https://tools.ietf.org/html/rfc6750#section-3.1">https://tools.ietf.org/html/rfc6750#section-3.1</a></li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> API Response, Authorization Response, Access Token Response</p>
<b>error_uri</b>	<p><b>URI de la page Web décrivant l'erreur</b> Paramètre optionnel présent lors d'une réponse en erreur. Il indique l'URL de la page Web de description de l'erreur.</p> <p><b>Cf. RFC correspondantes :</b></p> <ul style="list-style-type: none"> <li>- <a href="https://tools.ietf.org/html/rfc6749#page-45">https://tools.ietf.org/html/rfc6749#page-45</a></li> <li>- <a href="https://tools.ietf.org/html/rfc6750#section-3.1">https://tools.ietf.org/html/rfc6750#section-3.1</a></li> </ul> <p><b>Paramètre utilisé dans les requêtes suivantes :</b> API Response, Authorization Response, Access Token Response</p>

### 5.3.6 Authentification graduée

La notion d'authentification graduée vise à renforcer la sécurisation des accès aux ressources par les applications consommatrices en établissant une échelle de confiance entre les fournisseurs de ressources et le service d'authentification de l'utilisateur. Cette information se matérialise de deux façons :

- **Dans la preuve d'authentification :** définit le niveau de confiance attribué par le service d'authentification, en fonction du contexte d'authentification (durée, méthode utilisée...). D'une façon générale, elle représente la probabilité que l'utilisateur soit toujours en possession de son terminal (mobile, poste en agence...) pour limiter le risque d'usurpation.

- **Dans la ressource** : définit le niveau minimum de confiance requis pour accéder à la ressource, en fonction de l'analyse de risque métier effectué par le propriétaire de la ressource.

Ci-dessous les 4 niveaux d'authentification graduée, actuellement définis :

Description	
<b>0</b>	Authentification longue durée pour les accès non sensibles type accès smartwatch en lecture
<b>10</b>	L'utilisateur s'est authentifié depuis moins d'une semaine OU Echange entre serveurs, sans utilisateur
<b>20</b>	L'utilisateur s'est authentifié depuis moins d'une heure
<b>30</b>	L'utilisateur s'est authentifié fortement depuis moins d'une heure

### 5.4 SSO utilisateur et lancement d'API

La norme API définit les modalités d'exposition de ressources ou d'API et de consommation de celles-ci par une application consommatrice.

Concernant la fédération d'identité (SSO Utilisateur) entre IHM, il a été défini une norme IHM Groupe OpenID Connect [[NIHM](#)] qui remplace la précédente norme NNI IHM.

Les paragraphes ci-après décrivent quelques cas d'usage particuliers combinant la mise en œuvre d'une fédération d'identité, puis d'un appel d'API.

#### 5.4.1 Cas des applications accédées via jeton JWT (NNI IHM ou autres)

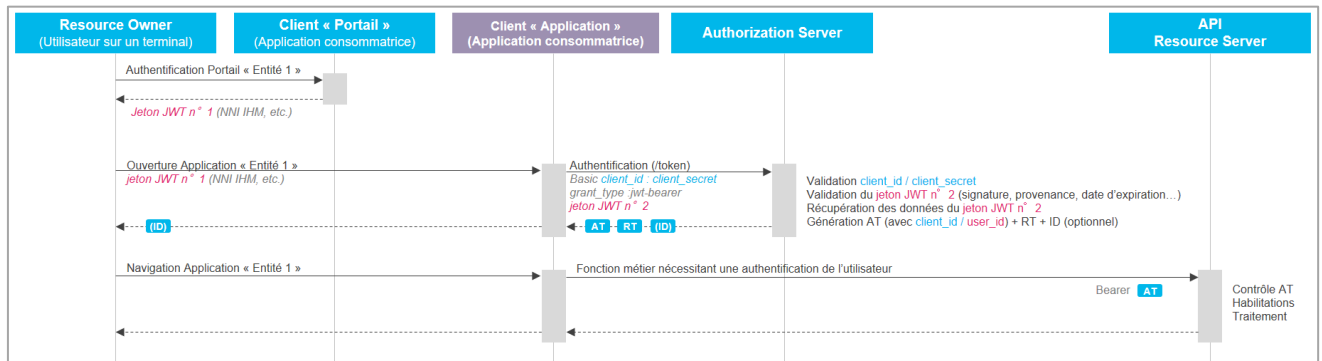
Cette cinématique permet à une application consommatrice d'obtenir des jetons OAuth2 / OpenID Connect lors de son lancement. Ces jetons sont obtenus à partir d'un jeton JWT n°2 généré par l'application consommatrice et contenant les données extraites du jeton JWT n°1 reçu (exemple jeton NNI IHM) lors de son lancement.

Cette solution repose sur les prérequis suivants :

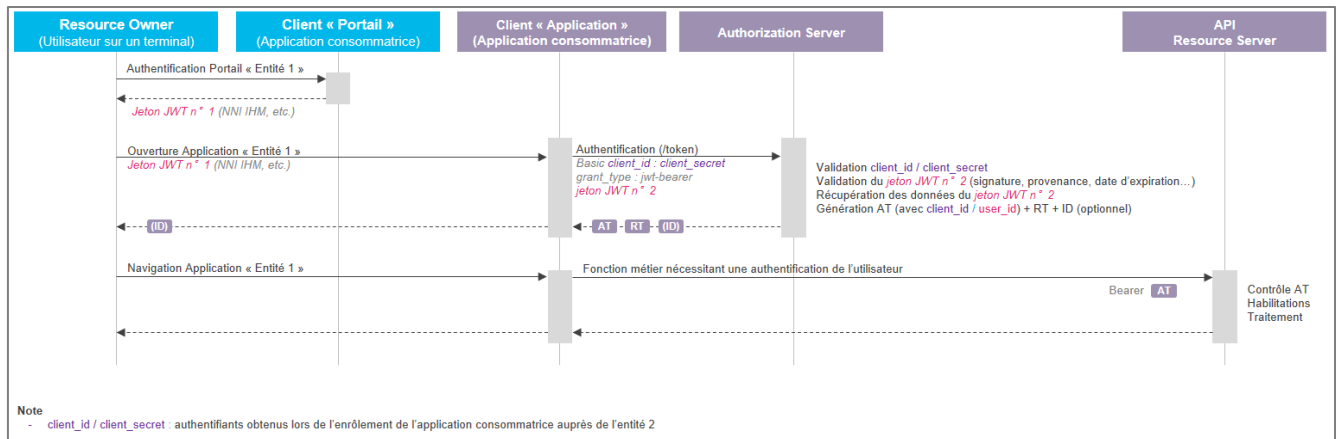
- Implémentation d'un grant\_type « urn:ietf:params:oauth:grant-type:jwt-bearer » ayant pour objectif de valider le jeton JWT n°2 présenté par l'application consommatrice. Ce grant\_type doit être implémenté sur l'autorization server de l'entité en charge de l'identification/authentification de l'utilisateur.
- Le jeton JWT n°2 doit répondre aux caractéristiques suivantes
  - Format JWT
  - Emission par l'application consommatrice qui devra être déclarée comme étant de confiance sur l'Authorization Server qui recevra le JWT.
  - Validité de la signature et du jeton (vérification de la date d'expiration)

- Présence de l'horodatage de la dernière authentification par login / mot de passe de l'utilisateur. Cette date est prise comme point de départ de la session d'authentification de l'autorization server (champs auth\_time des jetons d'accès, lien avec l'authentification graduée...)

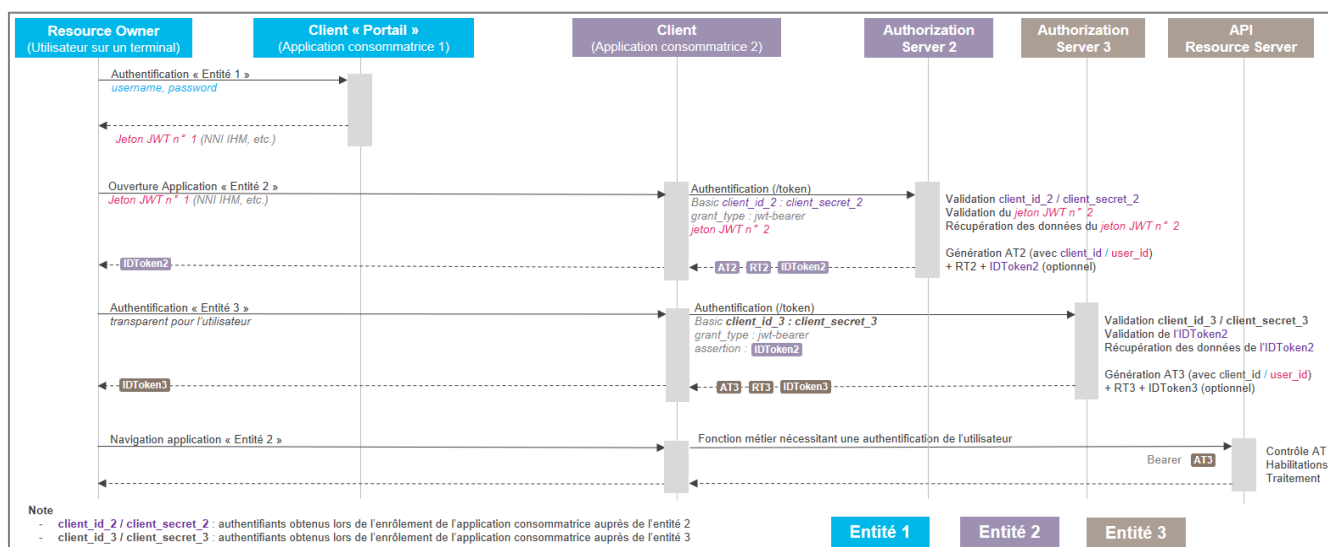
Le schéma ci-dessous illustre cette cinématique dans le cas d'usage UC 2-1 :



Le schéma ci-dessous illustre cette cinématique dans le cas d'usage UC 2-2 :



Le schéma ci-dessous illustre cette cinématique dans le cas d'usage UC 3-1 :



## Remarques :

- Le jeton NNI IHM n'est utilisé que pour initialiser ou renouveler la session d'authentification de l'utilisateur sur l'autorization server (équivalent à une authentification par login / mot de passe sur le mire d'authentification OpenId Connect).
- La durée de validité du jeton NNI IHM étant de quelques minutes, il doit être utilisé par l'applications consommatrice dès réception pour obtenir les jetons OAuth2 / OIDC.
- Dans le cas d'usage UC 2-2, avec 2 Entités appartenant au Groupe CA, après réception du jeton NNI IHM par l'application consommatrice de l'entité 2, la génération des jetons d'accès à l'API est de la responsabilité de l'Entité 2. La cinématique proposé ici côté Entité 2 n'est qu'une bonne pratique (non normatif). Le jeton NNI IHM permet d'effectuer une délégation d'authentification entre l'Entité qui a identifiée et authentifiée l'utilisateur et l'Entité propriétaire de l'application (consommatrice de l'API).
- Dans le cas d'usage UC 3-1, avec 3 Entités appartenant au Groupe CA, l'Authorization Server de l'Entité propriétaire de l'application consommatrice (mais pas de l'API, ni de l'utilisateur) peut produire les jetons d'accès à l'API. Il s'agit d'une adaptation du standard OAuth/OIDC au contexte Groupe (simplification). Le jeton NNI IHM permet d'effectuer une délégation d'authentification entre l'Entité qui a identifiée et authentifiée l'utilisateur et l'Entité propriétaire de l'application (consommatrice de l'API).
- Le renouvellement de jetons d'accès à l'API, par l'application consommatrice, s'effectue ensuite à partir du refresh token conformément au standard OAuth2.
- A noter, également, que si l'Authorization Server recevant l'ID Token est un Keycloak, le grant-type « jwt-bearer » ne sera pas accepté pour identifier l'utilisateur. En conséquence, dans ce cas d'usage (et uniquement, celui-ci), il faudra transmettre l'ID Token via la RFC8693 (OAuth 2.0 Token Exchange) et donc utiliser le grant-type « token-exchange ».



### 5.4.2 Cas d'usage d'appel d'une API depuis l'application cible

Ce cas d'usage est un exemple de combinaison entre les deux normes Groupe d'interopérabilité IHM et API. Dans une première étape, une application source (Web ou Mobile) lancera l'application cible (Web ou Mobile), puis l'application cible accèdera à une API de l'entreprise source pour répondre à un besoin fonctionnel particulier.

Ce cas d'usage se décline en deux cinématiques en fonction de type d'application : Web2Web et Mobile2Mobile.

Ces cinématiques sont détaillées dans la [Norme IHM Groupe](#), au paragraphe 3.5.8.

### 5.4.3 Cas d'usage d'appel d'une API depuis une autre API de l'entité cible

Ce cas d'usage est un exemple de combinaison entre les deux normes Groupe d'interopérabilité IHM et API. Dans une première étape, une application source (Web ou Mobile) lancera l'application cible (Web ou Mobile), puis l'application cible accèdera à l'une de ses API interne, et enfin, l'API interne de l'entité cible appellera une API de l'entreprise source.

Cette cinématique est détaillée dans la [Norme IHM Groupe](#), au paragraphe 3.5.9.

### 5.4.4 Cas d'usage de la transmission d'identité entre plusieurs API internes à une entité

Tout d'abord, il est à noter que la communication entre API internes à une entité ne relève pas de la présente norme API. En effet, la norme API standardise, uniquement, les échanges entre 2 entités ou entre une entité et un partenaire.

Par contre, la solution, ci-après, de propagation de l'identité d'un utilisateur au sein du SI interne d'une entité **constitue une bonne pratique** permettant de simplifier la mise en œuvre de cette propagation. A condition, bien entendu, que cette solution soit compatible avec l'analyse de risque de l'application concernée.

La solution est décrite en détail sur le site de l'OWASP (Open Web Application Security Project) qui, par ailleurs, préconise son usage : [OWASP Identity propagation patterns](#)

En synthèse, ce pattern de propagation d'identité distingue les accès externes (à l'entité) pour lesquels les cinématiques OAuth classiques seront utilisées, des accès entre API internes à l'entité (environnement de confiance) pour lesquels sera utilisé un JWT applicatif signé contenant les données d'identité de l'utilisateur et représentant une preuve de son identification. Ce JWT sera produit par un composant de confiance de l'entité et contiendra les données de l'Access Token externe, reçu en amont.

Ce pattern est, par exemple, utilisé par la société Netflix.

### 5.4.5 Autres cas possible de SSO entre applications via OpenID Connect

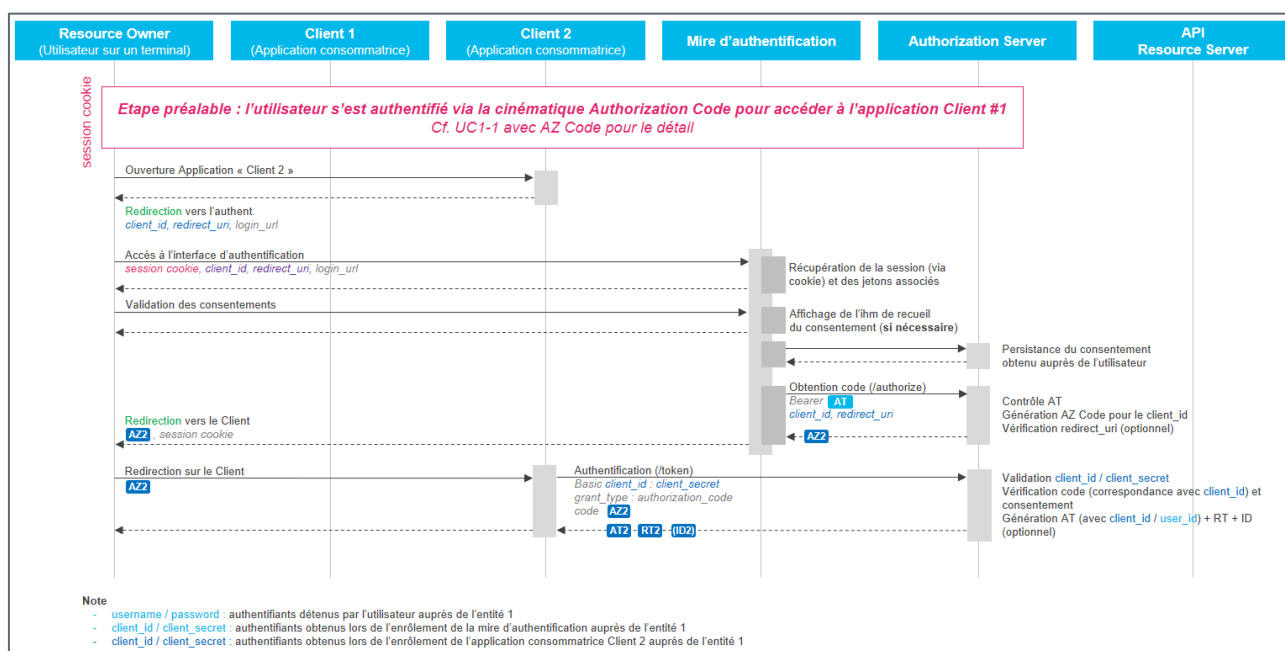
Conformément au standard OpenID Connect, l'adjonction d'une interface (IHM) d'authentification permet de maintenir une session utilisateur indépendamment de l'application consommatrice, et donc d'envisager des mécanismes de type SSO (Single Sign On) entre des applications web et mobiles. Pour les applications natives sur mobile, cette fonctionnalité repose sur la capacité de l'OS à partager le

cookie de session utilisateur entre application via (par exemple : Chrome Custom Tabs pour Android, SFSafariViewController pour iOS 9 ou 10, SFAuthenticationSession pour iOS 11...).

A noter que la fédération d'identité via le protocole OpenID Connect est le standard Groupe actuel. La spécification de ce standard est disponible sur l'Intranet AEG :

[Norme IHM Groupe OpenID Connect](#)

Pour exemple, le schéma ci-dessous illustre les capacités de SSO dans le cas d'usage UC1-1 avec une seconde application consommatrice :



**Remarque :** si l'access token détenu par la mire d'authentification et utilisé pour générer l'autorization code est expiré, celle-ci peut le renouveler à l'aide du refresh token obtenu lors de l'authentification de l'utilisateur. Cette étape n'est pas détaillée dans le schéma ci-dessus, se référer au chapitre 5.3.5.3.3.1 pour la cinématique associée.

## 6 Cas dérogatoires à l'utilisation de la spécification

### 6.1 Mise en œuvre de l'API DSP2

L'interface de l'API DSP2 est décrite par deux autres documents :

- Un document de spécification fonctionnelle qui indique les cas d'usage, les acteurs, et le fichier "Swagger" qui constitue le contrat d'interface technique pour le périmètre fonctionnel concerné
  - Lien : <https://www.stet.eu/en/psd2/>

**Les deux documents de description de l'API se suffisent à eux-mêmes et remplacent la spécification Norme API Groupe pour les API DSP2.** Ces spécifications sont conformes au Regulatory Technical Standards (RTS) de l'autorité bancaire européenne (EBA) qui s'imposent réglementairement aux banques européennes.

### 6.2 Mise en œuvre d'une API Webhook


Il existe un autre type d'API présente dans la norme d'échange par message asynchrone [[NASYNC](#)], il s'agit des **API Webhook**. Elles ne répondent pas à la même définition que l'API digitale.

Les API Webhook sont décrites au §4.3 de la norme d'échange par message asynchrone.

Ces API Webhook sont utilisées comme des URL de call back qu'un consommateur de message asynchrone indiquera lors de son enrôlement à un service d'envoi de notification envoyé en PUSH. La norme événement utilise ce pattern Webhook pour les échanges via Internet ou avec des partenaires.

## 7 Annexes

### 7.1 Documents de référence

Acronyme	Document de référence
ADSU	<p>Architecture de Développement Support de l'Urbanisme. Architecture applicative historique, s'appuyant sur http et XML et un protocole de sécurité des flux inter partenaires nommé IC04. Les spécifications peuvent être consultées à l'adresse suivante :</p> <p><a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/ADSU/Pages/adsu.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/ADSU/Pages/adsu.aspx</a></p>
ARAPI	<p>Analyse de risque MESARI des API Document sous l'égide du Comité Politiques et Standards SSI (anciennement nommé CPST) qui présente les risques associés aux échanges API et y associe une liste de mesures de sécurité à mettre en œuvre pour s'en prémunir :</p> <p><a href="https://secapi.adsi.credit-agricole.fr/lib/exe/fetch.php?media=prod-distrib:norme_api:analyse_de_risques_api_v1.2.pdf">https://secapi.adsi.credit-agricole.fr/lib/exe/fetch.php?media=prod-distrib:norme_api:analyse_de_risques_api_v1.2.pdf</a> (accessible aussi sur l'Intranet SECAPI)</p>
BPAPI	<p>Document décrivant les bonnes pratiques à utiliser pour définir et utiliser les ressources dans les API. Il est contenu dans l'archive qui contient le présent document. Il peut également être téléchargé sur le site de l'AEG à l'adresse suivante :</p> <p><a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAE_NormeAPI_BP_Ressources.pdf">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAE_NormeAPI_BP_Ressources.pdf</a> (accessible aussi cette <a href="#">page de l'Intranet AEG</a>)</p>
CAM0303	<p>Contenu du référentiel des identifiants CAM0303 actuels :</p> <p><a href="https://catalogue-referentiels.ca-sa.group.gca/8s_ihm/#/accueil/rechercheReferentielMaitre/afficheDonnee/CASA-R247">https://catalogue-referentiels.ca-sa.group.gca/8s_ihm/#/accueil/rechercheReferentielMaitre/afficheDonnee/CASA-R247</a></p> <p>➔ Cliquer sur l'icône « action »  du référentiel maître « TA NM303 »</p> <p>Documentation du format Groupe CAM0303 pour les UO Métier (i.e. identifiant d'entité)</p> <p><a href="https://service.ca-mocca.com/maia-collab/rec/docs/Unite%20Organisationnelle%20Metier%20-%20CAM0303-V1.pdf">https://service.ca-mocca.com/maia-collab/rec/docs/Unite%20Organisationnelle%20Metier%20-%20CAM0303-V1.pdf</a></p>
CANV	<p>Travaux concernant les canevas d'échanges dans le référentiel de l'AEG :</p> <p><a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Data/Historique/Canevas/Pages/Canevasdechange.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Data/Historique/Canevas/Pages/Canevasdechange.aspx</a></p>
CORS	<p>Cross Origin Ressource Sharing : spécification permettant côté client l'accès à une ressource depuis un domaine différent de son domaine d'origine.</p> <p><a href="http://www.w3.org/TR/cors/">http://www.w3.org/TR/cors/</a> Les navigateurs compatibles CORS (c'est le cas de tous les navigateurs récents) sont : <a href="http://caniuse.com/#feat=cors">http://caniuse.com/#feat=cors</a></p>

CSP	Content Security Policy : spécification permettant de déclarer un ensemble de restrictions à appliquer sur une ressource Web. <a href="https://www.w3.org/TR/CSP11/">https://www.w3.org/TR/CSP11/</a> Pour la mise en oeuvre, on peut se reporter au tutorial disponible à l'adresse : <a href="http://www.html5rocks.com/en/tutorials/security/content-security-policy/">http://www.html5rocks.com/en/tutorials/security/content-security-policy/</a>
DATE	Sous-ensemble des formats de dates décrits dans le standard ISO 8601 retenu par le W3C afin de limiter les erreurs et la complexité de gestion : <a href="http://www.w3.org/TR/NOTE-datetime">http://www.w3.org/TR/NOTE-datetime</a>
GLOSS_API	L'architecture d'entreprise Groupe a publié un glossaire API regroupant l'ensemble des définitions associées aux termes en lien avec le domaine des API : <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAE_glossaire_API.pdf">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAE_glossaire_API.pdf</a>
HAL	Hypertext Application Language est un standard qui définit des conventions pour exprimer les contrôles hypermédias, comme des liens, avec JSON. Il est en cours de validation à l'IETF et la version de travail des spécifications est disponible.
HSTS	http Strict Transport Security : spécification d'un mécanisme de politique de sécurité proposé pour HTTP, permettant à un serveur web de déclarer à un agent utilisateur (comme un navigateur web) compatible qu'il doit interagir avec lui en utilisant une connexion sécurisée (comme HTTPS). <a href="http://tools.ietf.org/html/rfc6797">http://tools.ietf.org/html/rfc6797</a>
IDHAB	Document de spécification des principes retenus pour la gestion des Identités et des habilitations : RefAEG_IdHab_Specifications_v2_2 disponible avec le présent document sur le site intranet de l'AEG à l'adresse : <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAEG_IdHab_Specifications_v2_2.zip">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI-API/RefAEG_IdHab_Specifications_v2_2.zip</a> (accessible aussi cette <a href="#">page de l'Intranet AEG</a> )
JOSE	JSON Object Signing and Encryption : ensemble de spécification décrivant une technologie utilisable pour signer et chiffrer des données structurées en utilisant le formalisme JSON. <a href="https://tools.ietf.org/html/rfc7165">https://tools.ietf.org/html/rfc7165</a>
JSON	Format d'échange de données textuelles standard, spécifié par la RFC 7159 de l'IETF <a href="https://tools.ietf.org/html/rfc7159">https://tools.ietf.org/html/rfc7159</a>
JWA	Catalogue des algorithmes utilisables dans le cadre des spécifications JSON Web Encryption (JWE) et JSON Web Signature (JWS): <a href="https://tools.ietf.org/html/rfc7518">https://tools.ietf.org/html/rfc7518</a>
JWE	Décrit la structure d'un jeton chiffré JSON Web Encryption (JWE) : <a href="https://tools.ietf.org/html/rfc7516">https://tools.ietf.org/html/rfc7516</a>
JWS	Décrit la structure d'un jeton signé JSON Web Signature (JWS) : <a href="https://tools.ietf.org/html/rfc7515">https://tools.ietf.org/html/rfc7515</a>
JWT	JSON Web Token (JWT) décrit un moyen de représenter des propriétés devant être transférées entre deux parties. Les propriétés sont codées sous la forme d'un objet JSON utilisé comme contenu utile d'un jeton signé JSON Web Signature (JWS) ou d'un jeton chiffré JSON Web Encryption (JWE)

	<a href="https://tools.ietf.org/html/rfc7519">https://tools.ietf.org/html/rfc7519</a>
NASYNC	Norme d'échange par message asynchrone (événements ou commandes) <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/Pages/NormeEVT.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/Pages/NormeEVT.aspx</a>
NIHM	Norme IHM Groupe OpenID Connect <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/Pages/NormeIHM.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/NormesEchanges/Pages/NormeIHM.aspx</a>
NHOT	RefAE_Norme_Nommage_URL_intranet.pdf Cette norme de nommage des URL Intranet du Groupe est de la responsabilité de l'Architecture d'Entreprise Groupe (CASA/ITD/CTO/AEG). Disponible sur l'Intranet AEG : <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Fabrication/Pages/NommageURL.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Fabrication/Pages/NommageURL.aspx</a>
REC	Référentiel des échanges : <a href="https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/REC/Pages/ReferentieldesEchanges.aspx">https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/REC/Pages/ReferentieldesEchanges.aspx</a>
REST	Style d'architecture décrit par Roy Fielding dans le document disponible à l'adresse <a href="https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm">https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm</a>
RMM	Modèle de maturité dans l'adoption des principes de REST, comprenant plusieurs niveaux allant de l'adoption minimum jusqu'à l'adoption complète, décrit par Leonard Richardson <a href="https://martinfowler.com/articles/richardsonMaturityModel.html">https://martinfowler.com/articles/richardsonMaturityModel.html</a>
SECAPI	Le standard SECAPI présente les activités, règles et mesures de sécurité applicative incontournables et obligatoires pour tous les projets (développements spécifiques ou acquisitions de progiciels). Le respect de ces règles et mesures conformes à l'état de l'art permet d'avoir une bonne prise en compte de la sécurité applicative. <a href="https://secapi.adsi.credit-agricole.fr/doku.php?id=Documentation">https://secapi.adsi.credit-agricole.fr/doku.php?id=Documentation</a>