h e g

Haute école de gestion
Genève

Sécurisation d'un réseau d'équipement IoT avec le protocole Oauth

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Sylvain MULLER

Conseiller au travail de Bachelor :

Ciaran BRYCE

Genève, 25 septembre 2018

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science en Informatique de gestion

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : http://www.urkund.fr/student_gorsahar.asp.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 25 septembre 2018

Sylvain MULLER

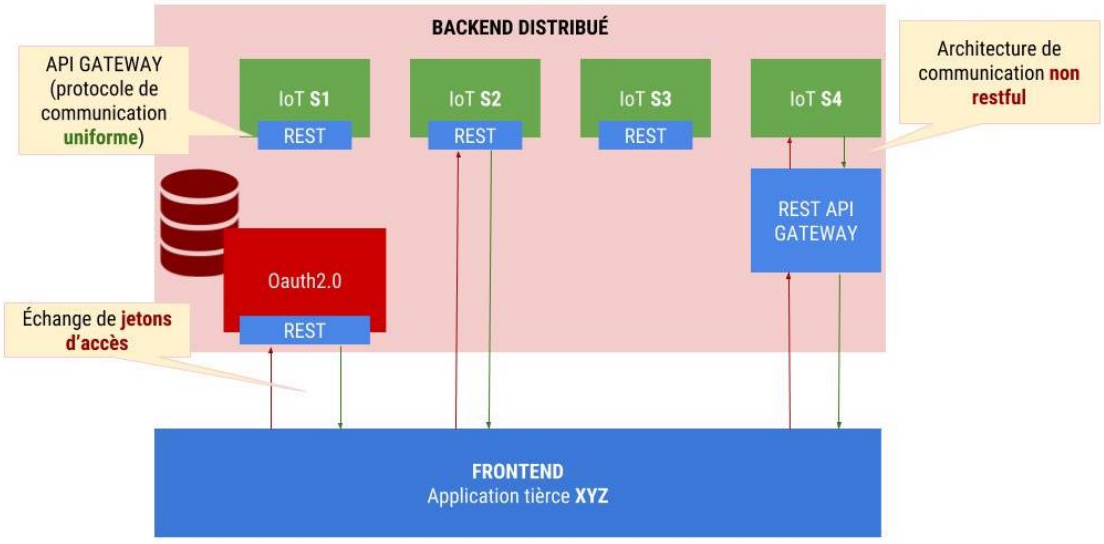
Remerciements

Je souhaite remercier mon directeur de mémoire, Monsieur Ciaran Bryce ainsi que Monsieur Gérard Ineichen pour leurs précieux conseils et pour les nombreuses heures de leur temps qu'ils m'ont accordé pour me soutenir et m'aider dans la réalisation de ce travail. Je tiens aussi à remercier ma famille et mes amis pour m'avoir épaulé durant ce travail de Bachelor et de manière générale, pour leur soutien inconditionnel durant toute la durée de mes études à la HEG.

Résumé

Selon une étude de l'entreprise américaine Gartner, le nombre d'objets (IoT) connectés à internet devrait presque doubler durant les deux prochaines années, passant de 11,1 milliards en 2018 à 20.4 milliards en 2020. La sécurité des IoT n'est pas facile à mettre en place. Certains appareils manquent simplement de puissance de calcul ou de mémoire et peuvent difficilement communiquer de façon sécurisée. D'autre part, une grande partie de l'internet des objets utilise la technologie « machine to machine » (M2M) pour communiquer entre eux et ne passe pas par un serveur centralisé. Le M2M nécessite un certain niveau de confiance entre les objets interconnectés qu'il est difficile de garantir même dans un réseau local. Dans le cadre de la sécurisation de systèmes IoT distribués, il est nécessaire de centraliser le système d'autorisation d'accès aux ressources et de pouvoir gérer et révoquer tout type de client au besoin.

Figure 1 – Réseau d'IoT sécurisé avec Oauth2.0



Le standard Oauth2.0 fourni les éléments nécessaires pour concevoir le système d'autorisation centralisé d'accès aux services distribués. Cependant, ce protocole est fortement dépendant de TLS. Toute communication doit être sécurisée au travers de requêtes https ce qui élimine de facto une partie non-négligeable des objets connectés actuellement sur le marché. Par ailleurs, en dérogeant légèrement au standard il est possible d'adapter le protocole pour supporter une partie des flux de communication sans TLS tout en offrant les mêmes garanties que dans sa forme initiale.

Table des matières

Liste des tableaux

Liste des figures

1. Introduction

Les cyberattaques contre les entreprises sont de plus en plus nombreuses et intenses. Les auteurs de ces attaques ne manquent pas d'inspiration pour trouver et exploiter la moindre vulnérabilité. Avec l'accroissement quasi exponentiel de l'internet des objets, la question inhérente à leur sécurité devient primordiale. En effet, les normes de sécurité pour les objets connectés ont tendance à être bien en retrait des standards acceptables actuels. Les attaques concernant les IoT sont aussi variées que nombreuses. Ransomware sur des télévisions connectées, prise de contrôle et détournement d'objets connectés et vols d'informations ne sont qu'une partie d'une liste qui s'agrandit d'année en année. Les fabricants d'IoT en tous genres ont tendance à faire la course aux parts de marchés et à l'innovation au détriment de la sécurité.

Une étude de Gartner parue en 2018 estime que 20% des entreprises dans le monde ont été témoins d'au moins une attaque basée sur l'internet des objets au cours des trois dernières années. Il estime aussi que de ce fait, le budget mondial alloué à la sécurité des IoT atteindra 1.5 milliard de dollars en 2018.

Les vulnérabilités directement liées à l'implémentation des objets connectés sont généralement des systèmes d'exploitation anciens, des logiciels non mis à jour, ou encore des identifiants de connexion codés en dur susceptibles d'être obtenus en faisant de la rétro-ingénierie[1]. En outre, les IoT constituent et constitueront dans les années qui vont suivre, d'immenses réseaux de ressources connectés accessibles et consommables au même titre que n'importe quel service web. Les enjeux sont doubles, puisque ce n'est pas seulement des standards de sécurité d'implémentation qu'il faut mettre en place mais c'est aussi une véritable stratégie de sécurité et de gestion des flux de communication. En effet, à l'heure actuelle, nos communications avec les IoT sont souvent de type « machine to machine » (M2M). Notre smartphone communique directement avec notre télévision connectée, notre smart-frigo ou encore notre véhicule sans passer par un parti dit de confiance. Chaque flux de communication avec un IoT part donc du principe que l'appareil avec lequel il communique est de confiance sans pouvoir réellement en établir la certitude. Seule l'intégrité du réseau permet de garantir à ces objets qu'aucun parti non-autorisé ne peut établir une connexion. Par ailleurs, comme mentionné ci-dessus, les objets connectés sont des ressources qu'on veut traiter

[1] En informatique, c'est le processus qui consiste à passer d'un exécutable au code source

comme n'importe quel autre service web distribué ce qui signifie souvent : fournir des api[2] à toute sorte de client.

Heureusement, il existe déjà un standard largement adopté dans le monde des services web et qui permet de répondre à ce besoin. C'est le « framework » Oauth2.0, un protocole de délégation d'autorisation. Il permet d'autoriser une application cliente (site web, application mobile, micro service, etc…) tierce (de consommation) d'utiliser les services web exposés par une ressource protégée (un fournisseur) pour le compte d'un utilisateur (propriétaire de la ressource).

1.1 Pourquoi Oauth2.0

Oauth2.0 est avant tout un protocole qui permet d'échanger des jetons d'accès afin d'obtenir des autorisations. L'objectif principal d'un jeton d'accès est de se substituer au traditionnel nom d'utilisateur et mot de passe et d'englober différentes autorisations compréhensibles par le serveur de ressource. C'est un standard qui est aujourd'hui adopté par les leaders mondiaux du web (Google, Facebook, Amazon, Twitter, etc…) et qui a largement fait ses preuves. L'ensemble des spécifications que propose le « framework » offre les outils nécessaires permettant de gérer et de sécuriser les communications dans un réseau de ressources protégées (nos IoT en l'occurrence).

[2] Interface de programmation

2. Objet connecté

2.1 L'internet des objets (IoT)

L'union internationale des télécommunications définit l'internet des objets comme étant *« une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérable*[3] *existante ou en évolution ».* Les IoT représentent pour certains la troisième évolution de l'internet : le Web 3.0. L'idée étant qu'il est possible de connecter au réseau des objets courants de la vie de tous les jours et de les doter d'une capacité analytique afin qu'ils puissent agir de manière autonome. Aujourd'hui, l'avènement des IoT s'opère surtout dans 3 domaines :

- La e-santé : cet ensemble de bracelets et autres montres connectés proposant de suivre et d'analyser les activités de la vie quotidienne.
- La domotique : la centralisation et l'automatisation de l'ensemble des systèmes physiques, électroniques ou informatiques utilisés dans le bâtiment pouvant aller du simple réfrigérateur personnel connecté jusqu'au système de ventilation complet d'une structure.
- Mesure de soi (quantified self) : regroupe tous les outils, notamment les objets connectés, permettant à chacun de mesurer, d'analyser et de partager ses données personnelles.

Les objets connectés utilisés dans le cadre de ce travail de Bachelor sont des contrôleurs pouvant accéder à internet et auxquels il est possible de connecter toutes sortes de capteurs. En fonction des capteurs utilisés et de ce qu'on fait des données récoltées, on obtient un IoT appartenant à un de ces trois domaines. En l'occurrence, les capteurs utilisés pour les différents prototypes d'objets connectés sont du domaine de la domotique.

J'ai à ma disposition une station météo complète avec capteur de température, pluviomètre, anémomètre[4] et girouette[5] ainsi qu'un capteur de mouvement.

2.2 Étude de faisabilité

La Haute École de Gestion de Genève met à ma disposition un certain nombre d'objets connectés me permettant de créer un réseau de divers capteurs sécurisés avec le

[3] Capacité d'un système à fonctionner avec d'autres produits ou systèmes existants ou futurs
[4] Appareil permettant de mesurer la pression ou la vitesse du vent
[5] Appareil permettant de mesurer la direction du vent

protocole Oauth2.0. Afin de déterminer si les objets prêtés peuvent être utilisés dans la réalisation de ce travail, j'effectue une étude de faisabilité pour chacun d'entre eux.

Cette analyse doit permettre de vérifier que l'appareil connecté supporte :

- Le protocole TLS ou une librairie de cryptographie AES (Le prototype du serveur Oauth2.0 ne supportera que cet algorithme de cryptage)
- Une librairie d'encodage/décodage en base64
- Suffisamment de mémoire et de puissance pour supporter la partie critique du protocole.

En outre, n'ayant pas le temps dans le cadre de ce Bachelor de développer des librairies supportant l'intégralité de l'architecture de communication REST, l'existence de librairies de type client/serveur RESTful est un avantage.

Finalement, n'étant pas familiarisé avec les langages de programmation des microcontrôleurs (C/C++), cette étude me permet d'apprendre leur syntaxe et les concepts de programmation système qui en découlent.

Les IoT à dispositions :

- Arduino UNO
- NodeMCU
- Raspberry PI model 3b+

2.2.1 Arduino UNO

L'Arduino UNO est une carte microcontrôleur open-source basée sur le microcontrôleur ATmega328 et développée par Arduino.cc. La carte possède 14 pins digitaux et 6 pins analogiques et peut être alimentée par un câble USB ou une batterie externe de 9 volt.

Spécifications supplémentaires :

- Mémoire flash : 32KB
- SRAM : 2KB
- EEPROM : 1KB
- Vitesse d'horloge : 16Mhz

Support des principales technologies requises :

Tableau 1 : Arduino UNO

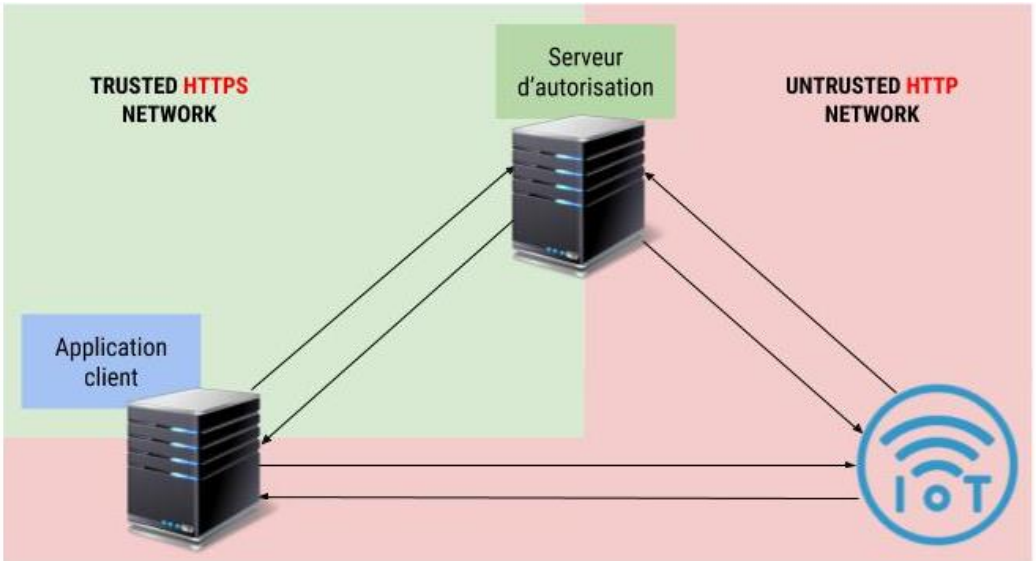
Technologie	Description
Support TLS/HTTPS	Aucun support TLS, la carte ne peut pas faire le SSL « handshake » impliqué à chaque requête. On ne peut ni exposer un serveur sécurisé, ni effectuer de requêtes HTTPS
Librairie AES	Il existe plusieurs librairies permettant d'utiliser les différents modes de cryptage et supportant plusieurs longueurs de clés (128, 192, 256).
Librairie serveur REST	Il existe une librairie orientée route pour l'Ethernet Shield[6] qui supporte une petite partie de l'architecture REST dont le dernier commit a été poussé en juillet 2015.
Librairie client REST	Il existe principalement une librairie connue pour l'Ethernet Shield qui supporte les principaux principes de l'architecture REST.
Connectique réseau	Contrôleur Ethernet W5100, 10/100Mb Power-over-Ethernet (PoE) IEEE802.3af

2.2.1.1 Support TLS :

L'Arduino UNO n'est pas assez puissant pour permettre de mettre en place un serveur HTTPS ou pour faire une requête sécurisé HTTPS. Ce point est primordial car l'objectif ici est de démontrer comment on peut établir une connexion encryptée entre deux appareils qui ne peuvent pas communiquer sous TLS. En l'absence de https, il est nécessaire d'encrypter directement les données qui entrent et qui sortent. La zone rouge de la Figure 2 représente les communications qui devront être cryptées avec AES.

[6] Une interface Ethernet destiné à l'Arduino UNO

Figure 2 : Arduino UNO - TLS



2.2.1.2 Mode de cryptage AES utilisé

En raison du faible espace mémoire disponible sur l'Arduino UNO, le mode et la taille de la clé de chiffrement relatifs au standard de cryptage avancé (AES) a dû être adapté en conséquence. En premier lieu, la clé de chiffrement utilisée est de 128bits, soit un mot de passe de 16 bytes et en second lieu, le mode choisi n'utilise pas de vecteur d'initialisation ce qui réduit encore la taille de mémoire à allouer. Bien que cette longueur de clé offre le niveau le plus bas de sécurité pour ce standard, il est tout de même à ce jour impossible de décrypter un « ciphertext » en un temps raisonnable.

2.2.1.3 Librairie REST

La librairie qui implémente en partie le standard REST permet de base d'exécuter un « callback » en fonction de l'url de la requête entrante et d'en extraire les paramètres sans traitement. En outre elle permet de retourner une réponse en format JSON avec un code HTTP 200. Définir plusieurs routes permettant d'envoyer des données provoque le redémarrage de l'appareil (problème lié au manque de mémoire du UNO). J'ai effectué un « fork » de la librairie de Bruno Luiz Silva afin de supporter certaines fonctionnalités requises dans le flux d'autorisation du protocole Oauth2.0.

Après modification de la librairie, les fonctionnalités suivantes ont été ajoutées/corrigées.

- Support de plusieurs callbacks avec des tableaux de taille réduite
- Un callback permettant de router les requêtes qui ne correspondent à aucun des critères définis (url, méthode), avec une erreur 404.
- Extraction et traitement séparé du corps de la requête, des « query params » et de l'en-tête http contenant le Bearer Token (le jeton d'accès utilisé par le protocole Oauth2.0).

- Support de plusieurs codes de réponses (200, 401, 404, 500).

Les paramètres d'allocation de mémoire en l'état actuel sont définis pour correspondre, « presque au byte près », à l'espace requis par une requête entrante du client, à savoir : une requête HTTP GET avec une url de petite taille et un token d'accès n'excédant par les 150 bytes, et c'est tout !

2.2.1.4 Conclusion de l'étude de faisabilité

Aux vues des points mentionnés ci-dessus, l'Arduino UNO ne semble pas le candidat idéal pour supporter le protocole à implémenter. La taille extrêmement réduite de mémoire disponible et l'absence de support TLS posent un véritable défi d'implémentation. Pour ces raisons, l'Arduino UNO peut être utilisé à titre de preuve de concept mais semble un choix peu judicieux dans des conditions d'utilisation réelles aux vues des alternatives qu'offre le marché. Le wavgat UNO, par exemple, est une copie de l'Arduino UNO offrant significativement plus de mémoire et représente une alternative viable.

2.2.2 NodeMCU

Le NodeMCU est une plateforme open-source comprenant un logiciel utilisant le langage de script Lua et est basé sur le microcontrôleur ESP-12. Il est développé et maintenu par ESP8266 Opensource Community. Il est aussi possible d'exécuter des programmes écrit en C et C++ grâce à un compilateur compatible avec le processeur et disponible via l'IDE d'Arduino. En outre, il possède 9 pins digitaux et un pin analogue.

Spécifications supplémentaires :

- Processeur : ESP8266
- Stockage : 4MB
- Mémoire : 128KB

Support des principales technologies requises :

Tableau 2 : NodeMCU

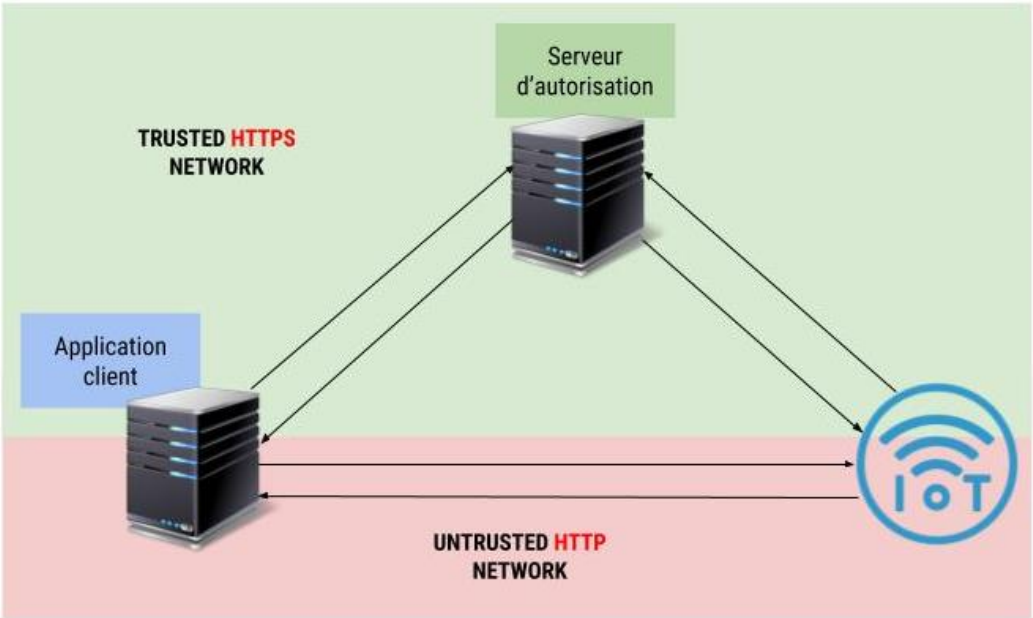
Technologie	Description
Support TLS/HTTPS	Support TLS client et serveur. En revanche, pour effectuer une requête https il faut renseigner l'empreinte numérique SHA1 du certificat du serveur cible.
Librairie AES	Il existe plusieurs librairies permettant d'utiliser les différents modes de cryptage et supportant plusieurs longueurs de clés.
Librairie serveur REST	La librairie officielle de l'ESP8266 permet de router et de traiter des requêtes http.

Librairie client REST	Il existe une librairie officielle pour l'ESP8266. En outre, d'autres librairies indépendantes sont activement maintenues.
Connectique réseau	Un module Wi-Fi IEEE 802.11 b/g/n compatible WEP ou WPA//WPA2

2.2.2.1 Support TLS :

Le NodeMCU est capable d'établir une connexion sécurisée avec ses clients. En outre, il peut effectuer des requêtes https, mais doit pouvoir valider le certificat via l'empreinte numérique SHA1. J'ai rencontré de nombreux problèmes pour effectuer une connexion https via un certificat qui n'est pas auto-généré. L'exemple fourni par la librairie officielle de l'ESP8266 est non-fonctionnel. De plus, il est impossible d'effectuer une vérification avec le certificat racine, ce qui poserait en production, d'énormes problèmes de maintenance (certificat valide de quelques semaines à quelques mois). Par conséquent, dans le cadre de ce travail de Bachelor, je n'effectue pas de vérification du certificat. Par ailleurs, mon objectif est de démontrer qu'il est possible de sécuriser un réseau d'internet des objets ayant des zones sans TLS. La communication entre le client et le NodeMCU est donc volontairement non sécurisée. La Figure 3 représente les communications qui devront être crypté avec AES.

Figure 3 – NodeMCU - TLS



2.2.2.2 Mode de cryptage AES utilisé

Le NodeMCU possède dix fois plus de mémoire que l'Arduino UNO. De ce fait, il est parfaitement capable de gérer des tableaux de caractères relativement grands. La

communication entre le client et la ressource protégée est cryptée au moyen du mode CBC et avec une clé de 256 bits. La combinaison de ce mode et de cette taille de clé est aujourd'hui considérée comme étant le meilleur moyen de crypter des données avec AES.

2.2.2.3 Librairie REST

La librairie officielle de l'ESP8266 ne permet pas de faire des requêtes https sans renseigner l'empreinte numérique SHA1. En raison des problèmes rencontrés avec cette vérification, j'utilise une autre librairie permettant de désactiver la dudit validation.

2.2.2.4 Conclusion de l'étude de faisabilité

Le NodeMCU est un nouveau contrôleur qui résout les principaux problèmes de sécurités de l'Arduino UNO quand il s'agit de l'exposer à travers un réseau internet. Au vu de son prix inférieur à 10 CHF, il est un candidat idéal pour connecter toutes sortes d'appareils et est donc un bon choix pour ce travail de Bachelor. A noter qu'il faudra tout de même implémenter une solution permettant de faire de la validation via un certificat racine avant qu'il soit envisageable de l'utiliser en production.

2.2.3 Raspberry Pi 3b+

Le Raspberry Pi modèle 3B+ est une évolution mineure du modèle 3B proposant une mise à jour du processeur et de la connectique réseau. En outre, il est équipé de 40 pins GPIO. Ce nano-ordinateur permet l'exécution de multiple système d'exploitation, dont la toute dernière version de Raspbian, une version de linux basée sur Debian spécialement optimisée pour les Raspberry Pi. Par ailleurs, il est fabriqué par la société Newark Corporation.

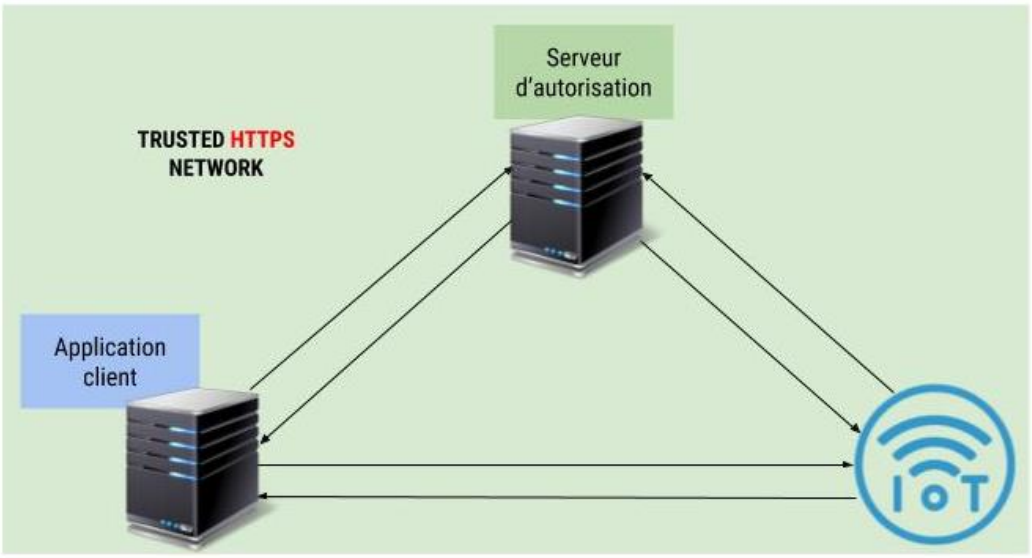
Spécifications supplémentaires :

- Processeur : Broadcom BCM2837B0 64 bit
- Stockage : Micro SD-CARD
- Mémoire : 1GB LPDDR2 SDRAM
- Connectique :
    - Wi-Fi IEEE 802.11 b/g/n/ac (2.4GHz et 5GHz)
    - Gigabit Ethernet, jusqu'à 300Mbps
    - Support Power-over-Ethernet (PoE)

Dans le cadre de ce travail de Bachelor, le modèle 3b+ exécute la toute dernière version de Raspbian. Il n'est pas nécessaire de présenter un tableau des technologies requises car le système d'exploitation (linux) de l'appareil nous permet de créer un environnement serveur complet et d'utiliser une multitude d'outils et de langages de programmation. En

outre, comme le démontre la Figure 4, le Raspberry Pi est un IoT idéal pour implémenter un flux de communication intégralement sécurisé avec TLS

Figure 4 : Communication sécurisée via https



2.2.3.1 Conclusion de l'étude de faisabilité

Le Raspberry Pi est le seul appareil à posséder de base tous les prérequis permettant d'établir un réseau d'IoT sécurisé avec le protocole Oauth2.0. Il est de facto utilisable dans le cadre de ce travail de Bachelor et constitue l'alternative étudiée la plus sûre pour une utilisation en production. C'est aussi l'IoT le plus chère avec un prix moyen de 50 CHF.

3. Choix d'implémentation

Cette section a pour objectif d'expliquer les choix qui ont été fait en matière d'implémentation des différents types de prototype servant à démontrer la thèse de ce travail de Bachelor. Ces choix concernent les langages utilisés pour programmer les prototypes ainsi que toutes autres technologies d'implémentation et sont présentés par catégorie.

3.1 Langages de programmations

3.1.1 C/C++

Le C et le C++ sont les langages de programmation système les plus utilisés au monde. L'Arduino UNO ne supporte pas d'autres langages de programmation. Le NodeMCU quant à lui, supporte en plus du C/C++, le Lua script. Afin de ne pas perdre trop de temps

à apprendre différent langage et étant de toute manière obligé de me familiariser avec le C et C++ , j'ai décidé de garder ces deux langages pour implémenter le NodeMCU.

3.1.2 PHP

Le nombre de langage connu permettant de faire du web se compte par dizaine. PHP est un langage conçu pour internet et qui offre un « time to market [7]» très court, notamment grâce aux nombreuses librairies de haut niveau qui permettent de s'abstraire de problèmes complexes. L'implémentation du serveur Oauth2.0 nécessitera de coder des processus relativement larges et complexes et d'utiliser pas mal de librairies de cryptographie. Pour les mêmes raisons que dans le point précédent, je choisi d'utiliser un langage que je maîtrise et qui me permet d'utiliser certaines librairies que je connais déjà bien.

3.2 Outils

3.2.1 SHELL

Afin d'automatiser le déploiement de l'environnement de développement du serveur Oauth2.0, j'utilise un script de type shell.

3.2.2 Make

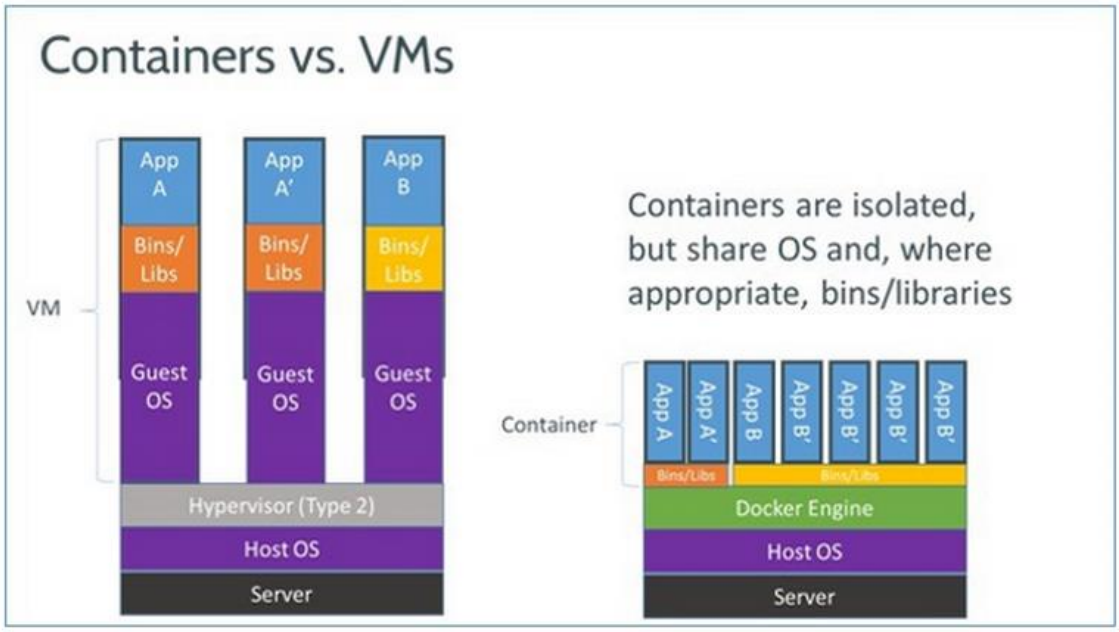
Toujours dans le but d'automatiser un certain nombre de processus relatifs au déploiement de l'application, j'utilise l'outil open-source « make » qui permet d'exécuter un certain nombre de tâches en fonction de dépendances.

[7] Le temps écoulé entre la conception et la vente d'un produit

3.2.3 Docker

Docker est un outil permettant d'empaqueter une application et ses dépendances dans un conteneur isolé. Elle pourra ensuite être exécutée sur n'importe quel serveur. Cette technologie permet d'améliorer, tout comme la virtualisation en général, la flexibilité et la portabilité d'exécution d'une application. Cependant, docker ne virtualise pas tout un système d'exploitation. Il se contente de partager les ressources système d'un même hôte entre différents conteneurs ce qui rend cette technologie bien moins gourmande et bien plus scalable. La Figure 5 schématise le fonctionnement de docker par rapport aux machines virtuelles.

Figure 5 : Conteneur vs machine virtuel



https://www.zdnet.com/article/vmware-buys-into-docker-containers

3.2.4 Docker-compose

Cet outil permet de connecter plusieurs conteneurs isolés à travers un réseau et d'orchestrer leur déploiement en commun sur un même hôte.

4. Le protocole Oauth2.0

Oauth2.0 est un protocole qui permet à une application tierce d'obtenir un accès limité auprès d'une ressource protégée. L'objectif est de garantir au propriétaire d'une ressource accessible via un service https, qu'aucun client ne pourra obtenir un accès auprès d'elle sans avoir été identifié et autorisé au préalable.

4.1 Internet Engineering Task Force (IETF)

L'IETF est un organisme de normalisation mondiale dépendant de l'Internet Society et qui a pour objectif d'élaborer et de promouvoir des standards internet, notamment les normes qui composent la suite des protocoles Internet (TCP/IP). Cet organisme publie des demandes de commentaires (request for comments, RFC) qui décrivent les aspects techniques d'Internet.

Le « framework » Oauth2.0 est standardisé par la RFC 6749. En outre, ce protocole utilise en interne d'autres standards Internet qui seront mentionnés et expliqués au cours de cette thèse.

4.2 Le framework d'autorisation Oauth2.0

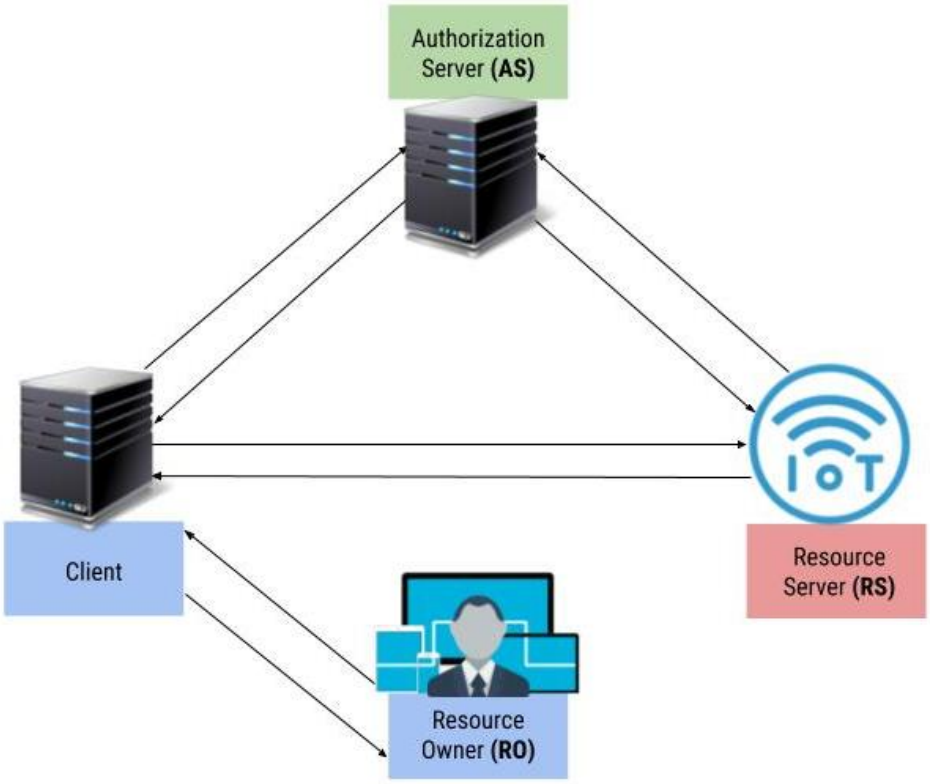
La RFC 6749 a pour objectif de standardiser les flux d'interactions qui régissent le protocole Oauth2.0. Cette spécification fait suite à la RFC 5849 qui décrit le protocole obsolète Oauth1.0.

4.2.1 Les rôles

Le protocole définit quatre rôles :

- **Le détenteur de la ressource (RO)**, plus communément appelé « resource owner » en anglais, est toute entité capable d'obtenir un accès auprès d'une ressource protégée. Il peut s'agir d'une application client comme d'une personne. Dans ce dernier cas on parle alors d'utilisateur final ou « end user ».
- **Le serveur de ressource (RS)** a pour responsabilité d'héberger une ressource protégée capable d'accepter et de répondre à des requêtes en utilisant des jetons d'accès. Dans le cadre de ce travail de Bachelor, les RS sont des objets connectés (IoT).
- **Le client** est une application effectuant des requêtes auprès d'une ressource protégée pour le compte du détenteur de la ressource. A noter que ce rôle ne définit absolument pas la nature de l'implémentation du client. Cette dernière peut aussi bien être un client standalone s'exécutant sur un ordinateur de bureau qu'une application mobile ou web.
- **Le serveur d'autorisation (AS)** a pour responsabilité d'émettre des jetons d'accès après avoir réussi à authentifier le détenteur de la ressource et obtenu, si besoin, les autorisations nécessaires auprès du détenteur la ressource.

Figure 6 : Les rôles



La Figure 6 représente les différents rôles du protocole Oauth2.0 ainsi que ses principaux flux de communication.

4.2.2 Le jeton d'accès

Un jeton d'accès est une chaîne de caractère émis par un serveur d'autorisation pour un client. Ce dernier peut présenter ce jeton d'accès à une ressource protégée et en obtenir l'accès. Cette chaîne de caractère est généralement, mais pas toujours, opaque pour le client. Elle représente et spécifie l'ensemble des autorisations que le détenteur de la ressource a accordé. L'objectif principal d'un jeton d'accès est de se substituer au traditionnel nom d'utilisateur et mot de passe et d'englober différentes autorisations compréhensibles par le serveur de ressource.

4.2.2.1 La famille des JOSE

On ne peut pas parler de jetons d'accès dans le cadre de Oauth2.0 sans aborder le sujet des « Javascript Object Signing and Encryption ». Les JOSE sont composés de deux protocoles aujourd'hui très largement repandus et sont standardisés par les RFC 7515 (JSON Web Signature - JWS) et 7516 (JSON Web Encryption - JWE). **Les spécifications JWS et JWE offrent une structure pour créer des JSON Web Token (JWT) encryptés ou signés.** Ce « token » est standardisé par la RFC 7519. Le JWT

ainsi que les caractéristiques qui le composent le rend particulièrement attrayant dans le flux d'interaction du protocole Oauth2.0 et est donc abondamment utilisé dans le cadre de ce travail de Bachelor. En outre, pour simplifier le développement et la gestion des clés de chiffrement, tous les JWE émis par le serveur d'autorisation prototype utilisent des algorithmes de cryptage symétrique.

Le JWT:

Un JSON Web Token est un jeton d'autorisation ou d'accès compact dont le corps est signé ou crypté et contient principalement des informations représentant la portée complète d'une autorisation (l'ensemble des paramètres qui définissent une autorisation). Il a la particularité d'être d'une part auto-explicatif et d'autre part d'être protégé contre toute tentative de modification grâce à une signature digitale.

Un JWT créer à partir de **la structure JWS** possède trois parties distinctes (le contenu de chacune de ces parties décrites ci-dessous peut être non exhaustif) :

- L'en-tête (header) : contient des informations sur le type de « token » et la signature utilisée (algorithme).
- Le contenu (payload) : contient des « claims ». Les claims standardisés sont principalement utilisés pour décrire la portée d'une autorisation (pas seulement les privilèges mais aussi pour qui le jeton est destiné, à quelle date il expire, qui l'a émis, etc…).
- Une signature : créée à partir de l'en-tête, du contenu et d'une clé secrète.

La Figure 7 représente un JWT signé.

Le JWT créer à partir **de la structure JWE** possède cinq parties distinctes (le contenu de chacune de ces parties décrites ci-dessous peut être non exhaustif) :

- L'en-tête (header) : contient des informations sur le type « token » et le type de cryptage utilisé (algorithme).
- Une clé de cryptage : une clé elle-même encryptée avec la clé publique du bénéficiaire.
- Le vecteur d'initialisation : le bloc de bits combiné avec le premier bloc de données.
- Le contenu : le texte encrypté (cipher text) contenant les claims.
- Le tag d'authentification : qui permet de prouver l'authenticité des données encryptée (MAC).

L'en-tête et le contenu du JWT sont encodés en base 64 afin que tous les caractères puissent être transportables au travers du protocole http puis ils sont mis bout-à-bout mais séparés par des points

Exemple d'un JWT créé à partir de la structure JWS et de l'algorithme de signature HS256 :

Figure 7 : JSON Web Token créé avec la structure JWS

La spécification concernant le JWT standardise un certain nombre de claims pouvant être utilisé pour décrire la portée d'autorisation d'accès du jeton. Ceux-ci sont notamment utile lorsque la ressource protégée ou le serveur d'autorisation souhaite vérifier la validité du jeton d'accès. Quand c'est le serveur qui porte cette responsabilité, on parle d'introspection de token « token instrospection », une spécification supplémentaire du protocole Oauth2.0 qui sera à l'étude dans le chapitre 4.2.8.

Les claims enregistrés :

- Iss (Issuer) : l'entité ayant émis le JWT, par exemple le serveur d'autorisation.
- Sub (Subject) : le sujet du JWT.
- Aud (Audience) : le bénéficiaire du JWT (la ressource protégée).
- Exp (Expiration time) : la date et l'heure après laquelle le JWT ne doit plus être accepté (format timestamp). Dans notre cas, le serveur d'autorisation devra s'assurer que le jeton d'accès n'est pas expiré lors du « token instrospeciton ».
- Nbf (Not before) : la date et l'heure avant laquelle le JWT ne doit pas être accepté (format timestamp).
- Iat (Issued at) : la date et l'heure à laquelle le JWT a été émis (format timestamp).
- Jti (JWT ID) : un identifiant unique qui assure que la probabilité que deux JWT similaire soit créé reste faible. Ce claim peut aussi être utilisé afin d'éviter que le JWT soit réutilisé (nonce).

Pour tous les claims indiquant un horaire (nbf, exp, iat), il est important que tous les services ayant la responsabilité de valider les jetons d'accès soient sur le même fuseau horaire que l'émetteur du dit jeton. Dans le cadre de ce travail de Bachelor, le serveur qui émet le jeton est aussi chargé de le valider, par conséquent le choix du fuseau n'a pas d'importance mais est réglé par convention sur GMT + 0.

4.2.2.2 Les JWK

Une JSON Web Key (JWK) est une structure de données JSON standardisée par la RFC 7517 qui représente une clé de cryptographie.

Figure 8 : Claims

La spécification prévoit un certain nombre de propriétés sous forme de méta-information qui ont pour objectif de décrire la clé de cryptographie.

Les paramètres enregistrés :

- Kty : la famille d'algorithme qui est utilisée. Par exemple, « oct » pour signifier symétrique.
- Use : l'objectif de l'utilisation de la clé. « sig » pour vérifier une signature ou « enc » pour encrypter des données.
- Key_ops : l'opération pour laquelle la clé a été créée (sign, verify, encrypt, etc…)
- Alg : l'algorithme à utiliser avec la clé
- Kid : un identifiant unique

Étant donné que le prototype du serveur d'autorisation utilise uniquement des clés symétriques, la liste ci-dessus est non-exhaustive. En effet, il existe d'autres paramètres standardisés à destination des clés publiques.

Dans le cadre de l'implémentation du prototype du serveur d'autorisation, d'autres propriétés non-standardisées sont utilisées.

Les paramètres non-enregistrés :

- K : la lettre « k » est utilisée par convention pour désigner la valeur d'une clé symétrique
- Enc : l'algorithme de cryptage de contenu utilisé dans le cadre de la création d'un JWE.

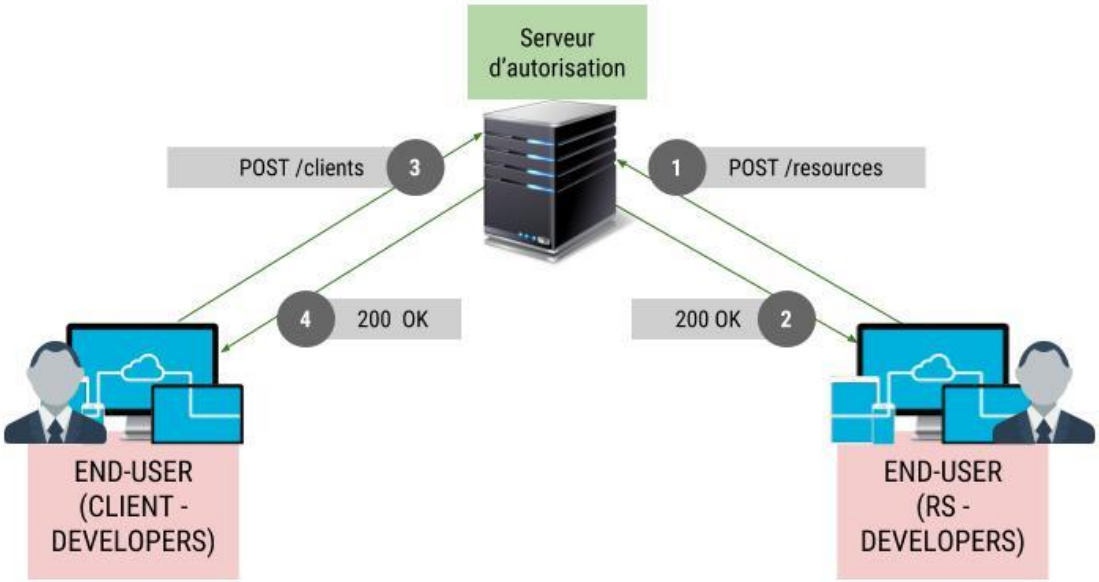
4.2.3 L'enregistrement

Avant d'expliquer et de décrire les processus de communication permettant à un client d'obtenir l'autorisation d'accéder à une ressource protégée, il est important de définir les étapes initiales nécessaires au bon fonctionnement du protocole.

Oauth2.0 demande que chaque client soit enregistré auprès du serveur d'autorisation. La façon dont le client est enregistré est en dehors du champ d'application du protocole mais c'est en général un utilisateur final qui procède à l'enregistrement de l'application via une page web.

En outre, les spécifications supplémentaires apportées par le « token introspection » dans le protocole Oauth2.0 demande que les ressources protégées soient aussi enregistrées.

Figure 9 : Enregistrement



4.2.3.1 Le client

Lors de l'enregistrement d'un nouveau client auprès du serveur d'autorisation, certaines informations devraient toujours être demandées par souci de sécurité.

- Le type de client
- Le/les url/s de redirection/s

Le type de client :

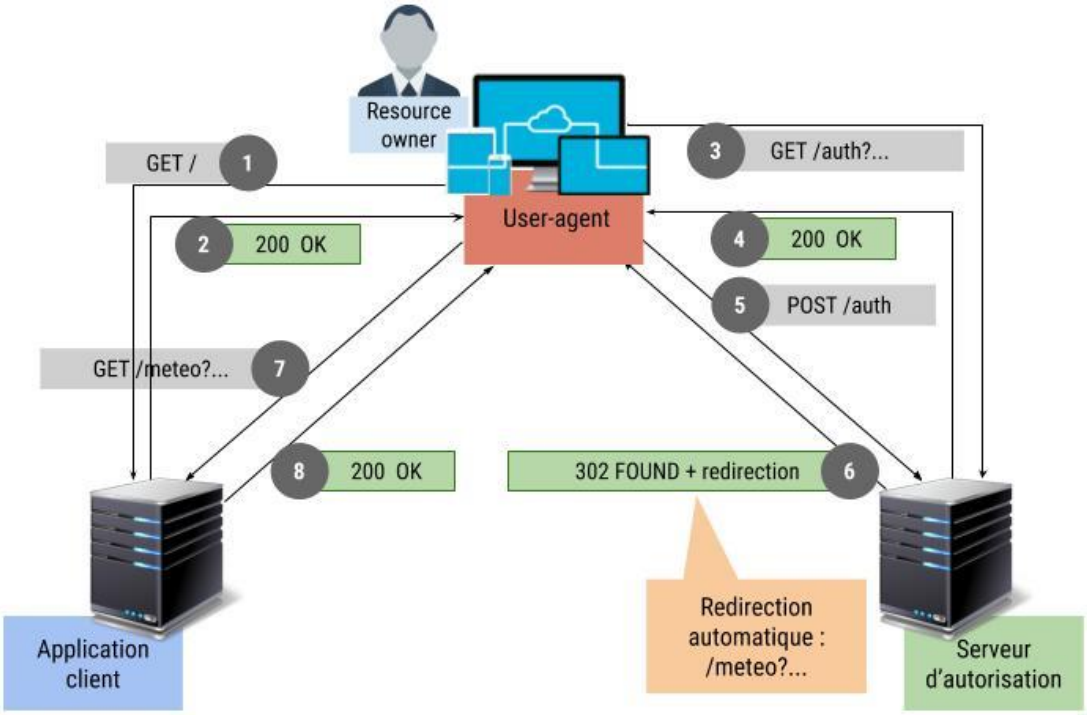
On distingue deux types de clients. Cette distinction dépend de la capacité du client à maintenir secret ses identifiants et mots de passe.

- Public : le client public est une application dont on ne peut garantir le maintien de la confidentialité de ses identifiants. Ce sont en général tout type d'application ne s'exécutant pas du côté d'un serveur et dont l'information sur les identifiants du client peut être acquise par un utilisateur de l'application (Par exemple : une application javascript s'exécutant sur un navigateur).
- Confidentiel : un client confidentiel est capable de maintenir ses identifiants de connexion privé. Ce sont typiquement des applications s'exécutant côté serveur.

Les URL de redirection :

Certains des flux d'obtention d'un jeton d'accès traités au chapitre 4.2.5 nécessitent que le détenteur de la ressource soit redirigé automatiquement vers une url au moment de l'obtention de l'autorisation d'accès. Afin qu'une application tierce puisse obtenir, par exemple, les privilèges nécessaires pour accéder à une ressource protégée détenue par son utilisateur courant, elle lui fournit un lien permettant de s'authentifier auprès du serveur d'autorisation. Si l'authentification réussi, le serveur d'autorisation redirigera alors automatiquement le détenteur de la ressource, via son agent utilisateur, sur une url appartenant au client.

Figure 10 : Redirection



La Figure 10 ci-dessus schématise le flux qu'un détenteur de ressource peut suivre avec son navigateur (user-agent). Lorsque l'utilisateur (resource owner) arrive sur la page d'accueil de l'application tièrce en point 2, il lui est alors proposé de s'authentifier par le biais d'une autre entitée (par exemple facebook). Généralement, cette proposition est présentée sous la forme d'un bouton (par exemple : « s'inscire avec facebook »). Si le détenteur de la ressource suit le lien, il attérit en point 4 sur une page html de connexion. Après s'être authentifié en point 5, le serveur d'autorisation redirige en point 6 et 7 le détenteur de la ressource vers une url qui lui permet de reprendre la navigation où elle en était avant son authentification.

Pour des raisons de sécurité et afin d'éviter qu'un attaquant ne puisse rediriger en point 7 un utilisateur sur une url qui est contrôlée par celui-ci, le serveur d'autorisation doit être en mesure de déterminer si l'url de redirection fournie par le client est connue et valide.

Création d'un nouveau client :

Dans le cadre de ce Bachelor, les attributs suivants doivent être renseignés pour enregistrer un nouveau client.

Tableau 3 : Requête création client

Requête : POST /clients		**Mediatype :** application/json
Paramètre	Exemple	Description
client_name	Meteo.ch	Nom qualificatif unique lisible par un humain
grant_type	Token	La méthode permettant d'obtenir un jeton d'accès. Dépend du type de client.
client_type	Public	Le type de client (public, confidentiel)
redirect_uri	https://meteo.ch/iot/1/temperature	Le ou les url/s de redirection
scope	ReadWeather	Le ou les privilèges de lecture et écriture

Le serveur d'autorisation confirme la création d'un nouveau client avec un code http 201 et une représentation complète du client nouvellement créé. En outre, le serveur génère automatiquement et attache à la réponse les informations suivantes :

Tableau 4 : Réponse création client

Réponse : POST /clients		**Mediatype :** application/json
Paramètre	Exemple	Description
client_identification	U42dfSgJ	Un identifiant unique généré aléatoirement qui permet au serveur d'autorisation de déterminer avec quel client il communique
client_password	W3l2WFQpYiFbLnR+M2Jjfg==	Un mot de passe fort de 16 bytes généré aléatoirement qui permet d'authentifier le client. Il est ensuite encodé en base 64 pour éliminer les caractères non supportés par le protocole http.
registration_date	12-09-2018 16:47:55	La date et l'heure de l'enregistrement.

4.2.3.2 La ressource

Une des spécifications supplémentaires que propose Oauth2.0 est l'introspection de jeton d'accès (token introspection) : un processus qui permet de valider un « token » et qui est l'objet du chapitre 4.2.8. Cette spécification supplémentaire est standardisée par la RFC7662. Conformément à la section 2.1 de la spécification, le serveur d'autorisation doit pouvoir déterminer identifier et authentifier la ressource protégée voulant valider le jeton d'accès.

Création d'une nouvelle ressource :

Dans le cadre de ce Bachelor, les informations suivantes doivent être renseignées pour enregistrer une nouvelle ressource. A noter que ce processus de création n'est pas implémenté dans le prototype et que toutes les ressources nécessaires à la démonstration existent déjà à la création de la base de données.

Tableau 5 : Requête création ressource

Réponse : POST /resources		**Mediatype :** application/json
Paramètre	Exemple	Description

resource_audience	Iot_a	Nom qualificatif unique lisible par un humain
resource_pop_method	Introspection	Le type de proof-of-possession (PoP) souhaité (standard, introspection). Ces méthodes sont décrites dans le chapitre 4.2.9.2.
key_size	16	Le taille de la clé de chiffrement symétrique partagée de (16, 24, 32)
shared_algorithm	AES128ECB	L'algorithme utilisé pour encrypter les communications entre le client et la ressource protégée
tls	0	La capacité de l'iot à communiquer de façon sécurisée
transmission_algorithm	AES128ECB	L'algorithme utilisé pour encrypter les communications entre le client et la ressource protégée

Le serveur d'autorisation confirme la création d'une nouvelle ressource avec un code http 201 et une représentation complète de la ressource nouvellement créée. En outre, le serveur génère automatiquement et attache à la réponse les informations suivantes :

Tableau 6 : Réponse création ressource

Réponse : POST /resources		**Mediatype :** application/json
Paramètre	Exemple	Description
resource_identification	HBrkK3zJ	Un identifiant unique généré aléatoirement qui permet au serveur d'autorisation de déterminer avec quelle ressource il communique
resource_secret	PzchKntkPWc4fUJUTWo3aA==	Un mot de passe fort de 16 bytes généré aléatoirement qui permet d'authentifier la ressource. Il est ensuite encodé en base 64 pour éliminer les caractères non supportés par le protocole http.
resource_registration_date	12-09-2018 16:47:55	La date et l'heure de l'enregistrement.

4.2.4 Authentification

Les ressources et les clients confidentiels doivent s'authentifier auprès du serveur d'autorisation à chaque requête. Sauf indications contraire du standard Oaut2.0, les clients et ressources doivent utiliser le schéma d'authentification « HTTP Basic » standardisé par la RFC 2617.

L'identifiant et mot de passe sont séparés par le caractère spécial « : » de la manière suivante :

user-pass : *identification : password*

- identification : l'identifiant du client ou de la ressource (chaine de caractère excluant « : »)
- password : le mot de passe du client ou de la ressource

Les informations d'authentification sont encodées en base 64 et transmises via un en-tête http de type autorisation :

Authorization : Basic MDEyMzQ1OnBhc3N3b3JkMTIzNDU=

En fonction de la méthode d'authentification négociée à l'enregistrement de l'entité (déterminée par le grant type et / ou la capacité à communiquer en TLS), seul l'identification ou le mot de passe peuvent être envoyés. Il incombe au serveur d'autorisation de déterminer quel en est la forme.

En principe, conformément au standard Oauth2.0, le serveur d'autorisation devrait accepter tous types de méthodes d'authentification. Dans le cadre de notre prototype, seule la méthode décrite ci-dessus est supportée, à savoir : « HTTP Basic »

4.2.5 Obtention d'un jeton d'accès

Oauth2.0 est avant tout un protocole qui permet à un client d'obtenir un jeton d'accès utilisable pour accéder à une ressource protégée. Plusieurs spécifications supplémentaires permettent de vérifier la provenance et la validité de celui-ci. Avant de détailler le fonctionnement de ces méthodes, nous nous intéressons aux différentes options proposées au client qui lui permettent d'obtenir une autorisation. Le standard décrit quatre méthodes d'obtention de jeton d'accès et leur utilisation dépend fortement du type de client. Chacun des processus d'obtention décrit ci-dessous suppose que le client ne possède encore aucun jeton d'accès et n'est pas en mesure d'en récupérer un nouveau par une voie facilitée (jeton d'accès de rafraichissement ou refresh token). Le chapitre 4.2.5.5 est consacré au client souhaitant remplacer un jeton d'accès généralement expiré ou invalide pour toute autre raison.

L'agent utilisateur (user-agent) :

Dans chacun des modèles ci-dessous, nous utilisons en plus des rôles décrits dans le sous chapitre 4.2.1 l'agent utilisateur. Bien que ce terme puisse désigner tous types

d'application cliente pouvant être à l'origine de requêtes http, nous limitons intentionnellement sa portée à tous types de navigateur web.

4.2.5.1 Client credentials

Un client a la possibilité d'obtenir un jeton d'accès en utilisant seulement ses propres identifications de connexion (client_identification, client_secret). Cette méthode ne peut être mise en œuvre pour des clients publics. En effet, le serveur d'autorisation doit pouvoir authentifier le client sans aucun doute et seuls les clients confidentiels peuvent le garantir. A noter que ce type d'accès n'est pas implémenté dans le prototype.

Figure 11 : Client credentials

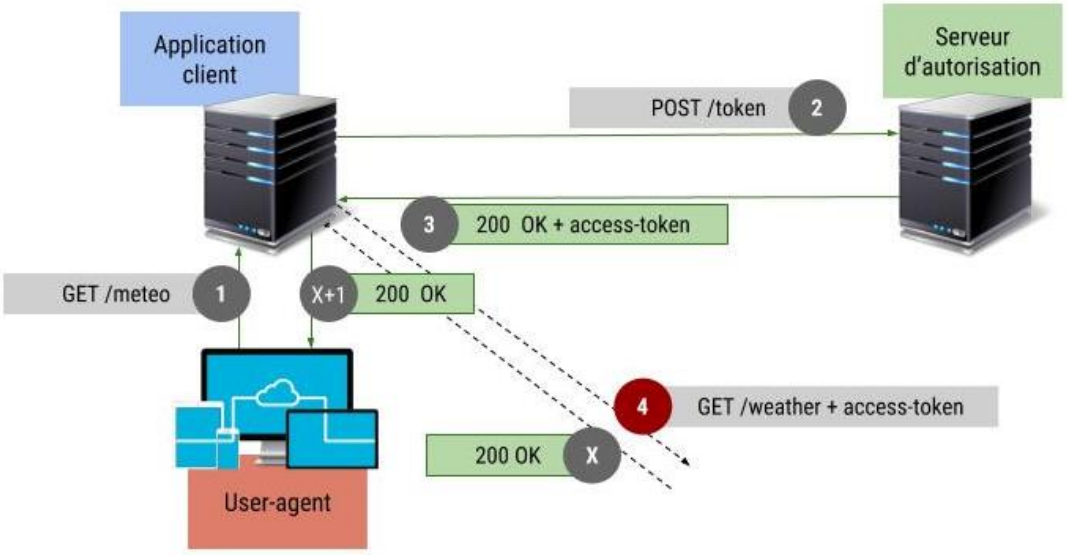


Tableau 7 : Flux de communication client credentials

1	Description	Quand le client ne possède pas déjà un jeton d'accès, le « user-agent » déclenche le processus d'obtention de l'autorisation en accédant au service http cible
	Exemple	GET /meteo
2	Description	Le client envoie une requête au serveur d'autorisation afin de récupérer un jeton d'accès. Il transmet ses identifiants de connexion afin de s'authentifier.
	Paramètre	**grant_type :** Requis : la valeur doit être « client_credentials » **scope :** Optionnel : les privilèges que le client souhait obtenir
	Exemple	**POST /token** Authorization : Basic AzHeaouvaRtjsahqASf https://oauth/token?grant_type=client_crendentials
3	Description	Le serveur d'autorisation authentifie le client et valide les privilèges demandés au besoin. Dans le cas favorable, il émet un jeton d'accès et répond avec un code http 200
	Paramètre	**access_token :** le jeton d'accès **token_type :** le type de jetons (jwt, jwe, pop) **expire_in :** le temps en secondes avant expiration

	Exemple	**200 OK**
		{
		"access_token":"eyJAfjskfIWERJFSsfaA…",
		"token_type":"JWT",
		"expires_in":10000
		}

4	Description	Le client effectue une requête auprès de la ressource protégée en y incluant le jeton d'accès. Ce processus fera l'objet d'une description détaillée dans le sous chapitre 4.2.7.
	Exemple	**GET /weather** Authorihzation : Bearer eyJAfjskfIWERJFSsfaA…

4.2.5.2 Resource Owner Password Credentials

Dans certain cas, le client peut obtenir un jeton d'accès en utilisant directement les identifiant du détenteur de la ressource. Ce type d'obtention s'opère uniquement quand le client et le détenteur de la ressource ont une relation hautement privilégiée et qu'aucune autre méthode ne peut être utilisée Contrairement au flux « client credentials », la ressource protégée n'est pas publique et appartient à un « resource owner » qui doit en autoriser l'accès. Étant donné que cette méthode utilise directement le mot de passe de l'utilisateur, le serveur d'autorisation doit être protégé contre les attaques de type « brute force ». A noter que ce type d'accès n'est pas implémenté dans le prototype et est de manière générale à proscrire.

Figure 12 : Resource owner password credentials

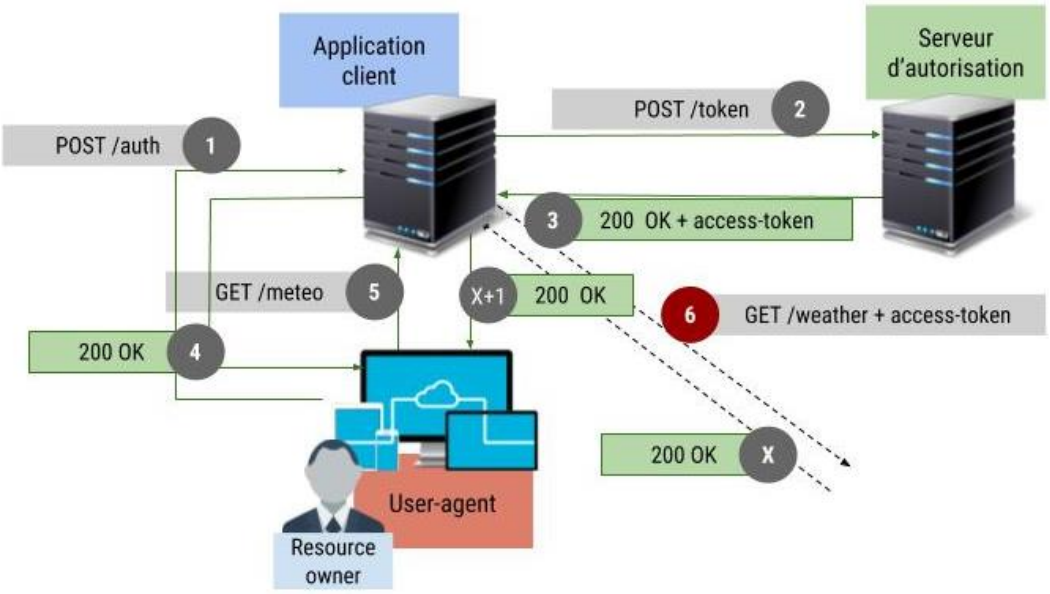


Tableau 8 : Flux de communication password credentials

1	Description	Quand le détenteur de la ressource n'est pas encore authentifié, il déclenche le processus d'obtention de l'autorisation via son « user-agent » en envoyant ses identifiants de connexion (généralement à travers un formulaire d'authentification).
	Exemple	**POST /auth** Authorihzation : Basic MDEyMzQ1OnBhc3N3b3JkMTIzNDU= OU username=johndoe&password=password123
2	Description	Le client envoi une requête au serveur d'autorisation afin de récupérer un jeton d'accès. Il transmet ses propres identifiants

		de connexion afin de s'authentifier ainsi que les identifiants du détenteur de la ressource.
	Paramètre	**grant_type :** Requis : la valeur doit être « password » **username :** Requis : l'identifiant du détenteur de la ressource **password :** Requis : le mot de passe du détenteur de la ressource **scope :** Optionnel : les privilèges que le client souhaite obtenir
	Exemple	**POST /token** Authorization : Basic AzHeaouvaRtjsahqASfiasfuU= https://oauth/token?grant_type=password&username=foo& password=bar
3	Description	Le serveur d'autorisation authentifie le client, le propriétaire de la ressource et valide les privilèges demandés au besoin. Dans le cas favorable, il émet un jeton d'accès et répond avec un code http 200.
	Paramètre	**access_token :** le jetons d'accès **token_type :** le type de jetons (jwt, jwe, pop) **expire_in :** le temps en secondes avant expiration **refresh_token :** le jeton de rafraichissement

	Exemple	**200 OK**
		Content-type : application/json
		{
		"access_token":"eyJAfjskfIWERJFSsfaA…",
		"refresh_token":" eyEHeridzhHkRUPJjdheuvoidf…",
		"token_type":"JWT",
		"expires_in":10000
		}

4	Description	Le client répond à la requête initialement déclenchée par le « user-agent » en affichant par exemple que l'authentification a réussi. Dans le cas favorable, il répond avec un code http 200.
	Exemple	**200 OK**
5	Description	Le « user-agent » déclenche le processus permettant de communiquer avec une ressource protégée en demandant au client par exemple d'afficher la page météo.

	Exemple	**GET /meteo**
6	Description	Le client effectue une requête auprès de la ressource protégée en y incluant le jeton d'accès. Ce processus fera l'objet d'une description détaillée dans le sous chapitre 4.2.7.
	Exemple	**GET /weather** Authorihzation : Bearer eyJAfjskflWERJFSsfaA…

4.2.5.3 Implicit Grant

Afin d'éviter que le client détienne à un moment donné les identifiants de connexion du détenteur de la ressource comme dans la méthode précédente, Oauth2.0 prévoit un type d'obtention d'accès de jeton fonctionnant avec un processus de redirection. Cette méthode permet de garantir au « resource owner » une confidentialité stricte de ses identifiants. En outre, ce flux est conçu pour les clients publics et ne prévoit pas de méthode simplifiée permettant de remplacer un jeton d'accès échu. Étant donné que le jeton d'accès est directement exposé à l'agent utilisateur du détenteur de la ressource, une application malveillante, par exemple une extension vérolée sur un navigateur pourrait obtenir une copie de l'autorisation. Le prototype du serveur d'autorisation implémente « Implicit Grant ».

Figure 13 : Implicit grant

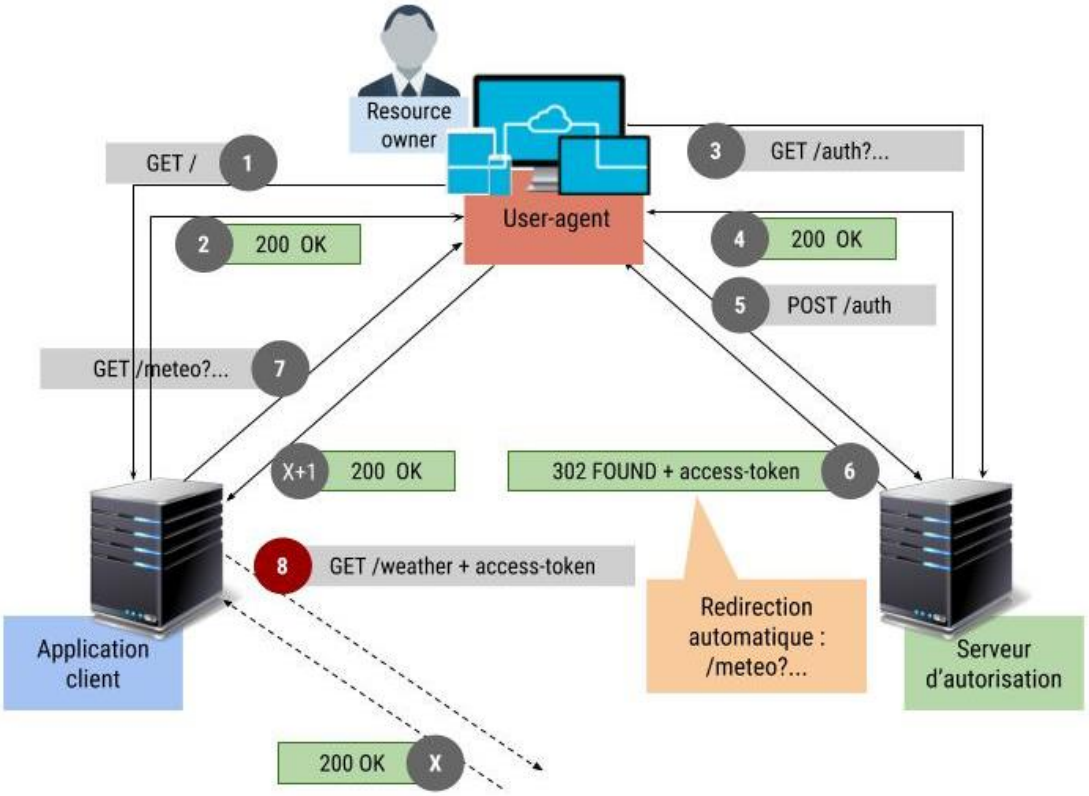


Tableau 9 : Flux de communication implicit grant

1	Description	Le détenteur de la ressource protégée envoie une requête d'accès au client via son « user-agent »
	Exemple	**GET /**
2	Description	Quand le détenteur de la ressource n'est pas encore authentifié, il répond avec un code http 200 et génère et affiche un lien de redirection vers le serveur d'autorisation (généralement sous la forme d'un bouton html).
	Exemple	**200 OK** Content-type : text/html
3	Description	Le détenteur de la ressource envoie une requête paramétrée au serveur d'autorisation via son « user-agent » afin d'obtenir une page de connexion.
	Paramètre	**response_type :** Requis : la valeur doit être « token » **client_id :** Requis : l'identifiant unique du client **redirect_uri :**

		Optionnel : une url de redirection valide (préalablement enregistrée) **scope :** Optionnel : les privilèges que le client souhaite obtenir **state :** Recommandé : une chaîne de caractère permettant de se protéger contre les attaques de type « cross-site request forgery (CSRF) »
	Exemple	**GET /auth ?...** https://oauth/auth?response_type=token&client_id=5HvdB43& redirect_uri=https://meteo.ch/meteo&scope=Rweather& state=aUewnvl45dJi
4	Description	Le serveur d'autorisation valide les paramètres de la requête. Dans le cas favorable, il répond avec un code http 200 et affiche une page de login
	Exemple	**200 OK** Content-type : text/html
5	Description	Le propriétaire de la ressource rempli le formulaire de l'étape précédente et déclenche via son « user-agent » le processus étendu d'obtention du jeton d'accès.
	Paramètre	**username :** Requis : l'identifiant du détenteur de la ressource **password :** Requis : le mot de passe du détenteur de la ressource **scope :** Optionnel : les privilèges que le détenteur de la ressource souhaite accorder à l'application tierce
	Exemple	**POST /token** username=foo&password=bar&scope=Rweather
6	Description	Le serveur d'autorisation authentifie le propriétaire de la ressource. Dans le cas favorable, il émet un jeton d'accès et répond avec un code http 302 et un en-tête de redirection automatique.
	Paramètre	**access_token :** Requis : le JWT émis par le serveur d'autorisation **token_type :** Requis : le type de jeton d'accès **scope :** Optionnel : les privilèges accordés par le détenteur de la ressource (peuvent différer de ceux demandé par le client) **state :**

		Requis :si le client a fourni la valeur « state », le serveur doit inclure la valeur exactement identique dans la réponse
	Paramètre non-standardisé	**shared_key :** Optionnel : une clé de chiffrement symétrique sans caractères spéciaux partagée entre le client et la ressource protégée qui permettra de crypter les données transmises entre le client et la ressource protégée ou d'établir la preuve de la possession. Ces points sont respectivement traités dans les chapitres 4.2.5.6 et 4.2.9.
	Exemple	**302 FOUND** Location : https://meteo.ch/meteo?access_token=eykjHZUFEWdjhwo89eIER&token_type=jwt&scope=Rweather&state=aUewnvl45dJi&shared_key=AghDF89Wr9Wfwer9
7	Description	Le « user-agent » redirige automatiquement le propriétaire de la ressource avec l'url fourni dans l'en-tête « location » de la réponse précédente.
	Paramètre	Le même paramètre que dans le point 6
	Exemple	**GET /meteo?...** https://meteo.ch/meteo?access_token=eykjHZUFEWdjhwo89eIER&token_type=jwt&scope=Rweather&state=aUewnvl45dJi&shared_key=AghDF89Wr9Wfwer9
8	Description	Le client effectue une requête auprès de la ressource protégée en y incluant le jeton d'accès récupéré dans l'url de la requête précédente. Au besoin, il encrypte l'information qu'il souhaite envoyer à l'aide de la clé partagée. Ce processus fera l'objet d'une description détaillée dans les sous chapitres 4.2.5.6 et 4.2.7.
	Exemple	**GET /weather** Authorihzation : Bearer eyJAfjskfIWERJFSsfaA…

4.2.5.4 Authorization Code Grant

Cette méthode d'obtention de jeton d'accès possède les mêmes avantages que le processus précédent (le client n'a jamais les identifiants du détenteur de la ressource), à ceci près qu'il est conçu pour des clients confidentiels et permet d'acquérir en plus, un jeton de rafraichissement (refresh token). En outre, il offre une couche de sécurité supplémentaire puisque le jeton d'accès n'est jamais directement exposé à l'agent

utilisateur. Le prototype du serveur d'autorisation implémente « Authorization Code Grant ».

Figure 14 : Authorization Code Grant

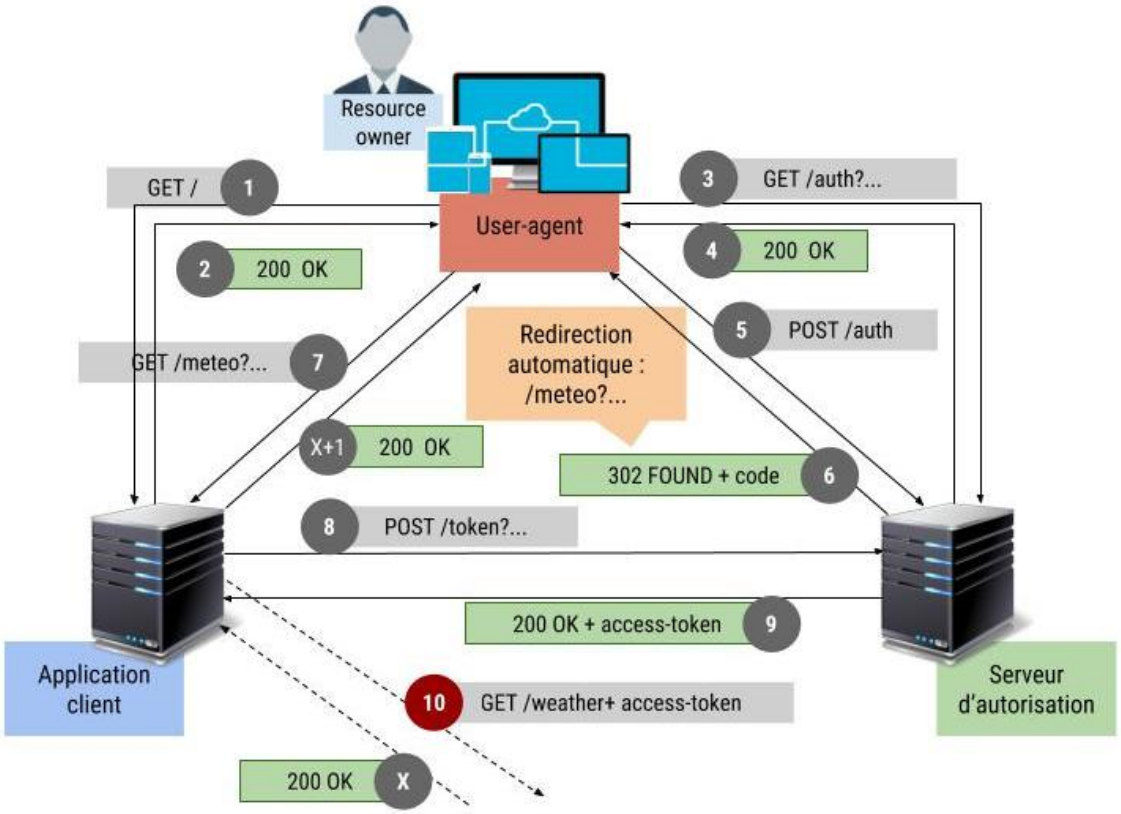


Tableau 10 : Flux de communication authorization code grant

1	Description	Le détenteur de la ressource protégée envoie une requête d'accès au client via son « user-agent ». Le début du processus est sensiblement identique aux flux « implicit » du chapitre 4.2.5.3.
	Exemple	**GET /**
2	Description	Quand le détenteur de la ressource n'est pas encore authentifié, il répond avec un code http 200 et génère et affiche un lien de redirection vers le serveur d'autorisation (généralement sous la forme d'un bouton html).
	Exemple	**200 OK** Content-type : text/html
3	Description	Le détenteur de la ressource envoie une requête paramétrée au serveur d'autorisation via son « user-agent » afin d'obtenir une page de connexion.

	Paramètre	**response_type :** Requis : la valeur doit être « code » **client_id :** Requis : l'identifiant unique du client **redirect_uri :** Optionnel : une url de redirection valide (préalablement enregistrée) **scope :** Optionnel : les privilèges que le client souhaite obtenir **state :** Recommandé : une chaîne de caractère permettant de se protéger contre les attaques de type « cross-site request forgery (CSRF) »
	Exemple	**GET /auth ?...** https://oauth/auth?response_type=code&client_id=5HvdB43& redirect_uri=https://meteo.ch/meteo&scope=Rweather& state=aUewnvl45dJi
4	Description	Le serveur d'autorisation valide les paramètres de la requête. Dans le cas favorable, il répond avec un code http 200 et affiche une page de login
	Exemple	**200 OK** Content-type : text/html
5	Description	Le propriétaire de la ressource rempli le formulaire de l'étape précédente et déclenche via son « user-agent » le processus étendu d'obtention du jeton d'accès.
	Paramètre	**username :** Requis : l'identifiant du détenteur de la ressource **password :** Requis : le mot de passe du détenteur de la ressource **scope :** Optionnel : les privilèges que le détenteur de la ressource souhaite accorder à l'application tierce
	Exemple	**POST /token** username=foo&password=bar&scope=Rweather
6	Description	Le serveur d'autorisation authentifie le propriétaire de la ressource. Dans le cas favorable, il émet un code d'autorisation et répond avec un code http 302 et un en-tête de redirection automatique.
	Paramètre	**code :** Requis : le code d'autorisation généré par le serveur d'autorisation permettant de récupérer le jeton d'accès

		state : Requis :si le client a fourni la valeur « state », le serveur doit inclure la valeur exactement identique dans la réponse
	Exemple	**302 FOUND** Location : https://meteo.ch/meteo?code=ey98zuGEZ7uGZu ow8jnfJ&state=aUewnvl45dJi
7	Description	Le « user-agent » redirige automatiquement le propriétaire de la ressource avec l'url fourni dans l'en-tête « location » de la réponse précédente.
	Paramètre	Le même paramètre que dans le point 6
	Exemple	**GET /meteo?...** https://meteo.ch/meteo?code=ey98zuGEZ7uGZuow8jnfJ &state=aUewnvl45dJi
8	Description	Le client envoie une requête au serveur d'autorisation en incluant ses identifiants de connexion dans le but d'obtenir un jeton d'accès. A partir de ce point, les étapes diffèrent du flux « implicit ».
	Paramètre	**client_id :** Requis : l'identifiant unique du client **code :** Requis : le code d'autorisation émis par le serveur d'autorisation **redirect_uri :** Requis : Si une url de redirection est fournie la requête d'autorisation (étape 3), le client doit fournir la même url en paramètre **grant_type :** Requis : la valeur doit être « authorization_code »
	Exemple	**POST /token?...** Authorization : Basic AzHeaouvaRtjsahqASf https://oauth/token?grant_type=authorization_code &code=ey98zuGEZ7uGZuow8jnfJ&state=aUewnvl45dJi &redirect_uri=https://meteo.ch/meteo
9	Description	Le serveur d'autorisation authentifie le client et valide le code d'autorisation. Dans le cas favorable, il émet un jeton d'accès et répond avec un code http 200.
	Paramètre	**access_token :** le jetons d'accès **token_type :** le type de jetons (jwt, jwe, pop) **expire_in :** le temps en seconde avant expiration **refresh_token :** le jeton de rafraichissement

	Paramètre non-standardisé	**shared_key :** Optionnel : une clé de chiffrement symétrique sans caractères spéciaux partagée entre le client et la ressource protégée qui permettra de crypter les données transmises entre le client et la ressource protégée ou d'établir la preuve de la possession. Ces points sont respectivement traités dans les chapitres 4.2.5.6 et 4.2.9.

	Exemple	**200 OK**
		Content-type : application/json
		{
		"access_token":"eyJAfjskfIWERJFSsfaA…",
		"refresh_token":" eyEHeridzhHkRUPJjdheuvoidf…",
		"token_type":"JWT",
		"expires_in":10000,
		……"shared_key":"AghDF89Wr9Wfwer9"
		}

10	Description	Le client effectue une requête auprès de la ressource protégée en y incluant le jeton d'accès. Au besoin, il encrypte l'information qu'il souhaite envoyer à l'aide de la clé partagée. Ce processus fera l'objet d'une description détaillée dans les sous chapitres 4.2.5.6 et 4.2.7.
	Exemple	**GET /weather** Authorihzation : Bearer eyJAfjskfIWERJFSsfaA…

4.2.5.5 Remplacer un jeton d'accès

Les processus « Authoriation Code Grant » et « Resource Owner Password Credentials » prévoient l'émission de jeton de rafraichissement (refresh token) par le serveur d'autorisation. Ce processus permet notamment à un client dont le « resource owner » est déjà authentifier d'obtenir un nouveau jeton d'accès de manière simplifiée. Le détenteur de la ressource n'aura donc pas à s'authentifier à chaque fois que le jeton d'accès expire et permet par conséquent au client de maintenir pour lui indéfiniment la connexion auprès de la ressource protégée.

Figure 15 : Refresh token

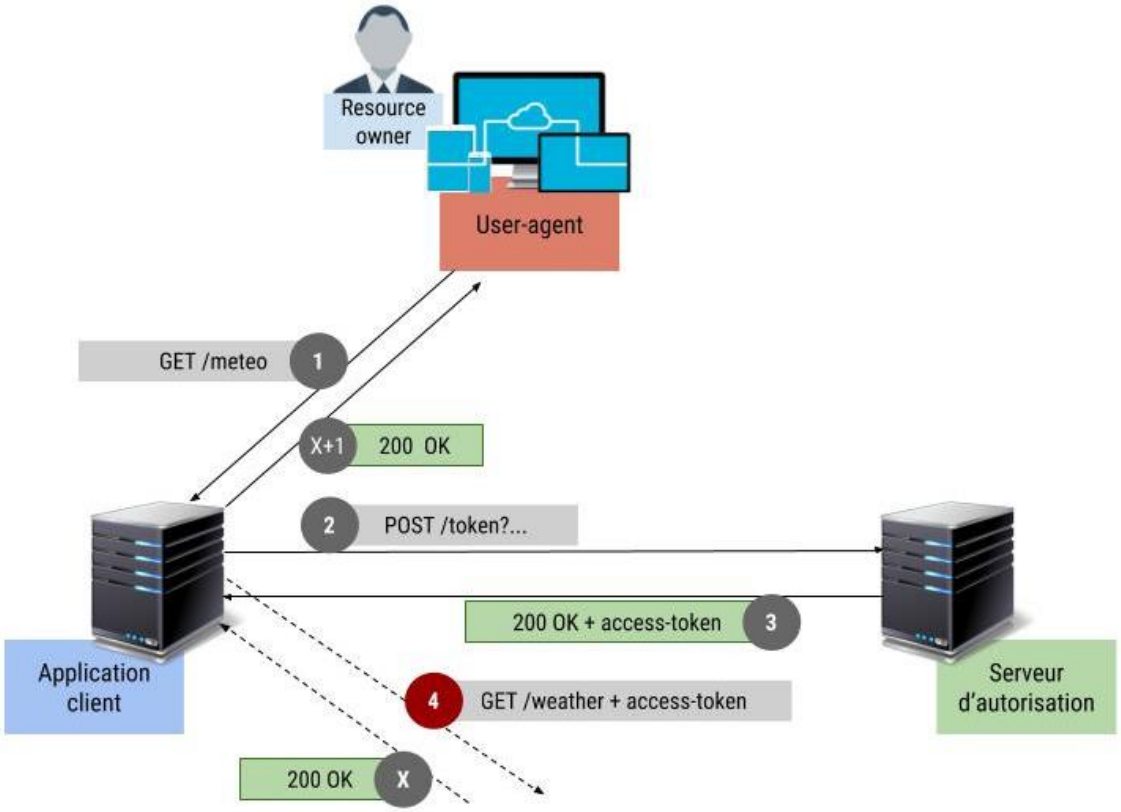


Tableau 11 : Flux de communication refresh token

1	Description	Quand le client possède un jeton d'accès expiré ou invalide, le « user-agent » déclenche le processus d'obtention de l'autorisation en accédant au service http cible
	Exemple	**GET /meteo**
2	Description	Le client envoie une requête au serveur d'autorisation en incluant ses identifiants de connexion dans le but d'obtenir un jeton d'accès
	Paramètre	**grant_type :** Requis : la valeur doit être « refresh_token » **refresh_token :** Requis : le jeton de rafraichissement émis par le serveur d'autorisation **scope :** Optionnel : les privilèges initialement accordés par le détenteur de la ressource.
	Exemple	**POST /token?...** Authorization : Basic AzHeaouvaRtjsahqASf https://oauth/token?grant_type=refresh_token

		&refresh_token= eyEHeridzhHkRUPJjdheuvoidf &scope=Rweather
3	Description	Le serveur d'autorisation authentifie le client et valide le jeton de rafraichissement. Dans le cas favorable, il émet un jeton d'accès et répond avec un code http 200.
	Paramètre	**access_token :** le jetons d'accès **token_type :** le type de jetons (jwt, jwe, pop) **expire_in :** le temps en seconde avant expiration **refresh_token :** le jeton de rafraichissement
	Paramètre non-standardisé	**shared_key :** Optionnel : une clé de chiffrement symétrique sans caractères spéciaux partagée entre le client et la ressource protégée qui permettra de crypter les données transmises entre le client et la ressource protégée ou d'établir la preuve de la possession. Ces points sont respectivement traités dans les chapitres 4.2.5.6 et 4.2.9.

	Exemple	**200 OK**
		Content-type : application/json
		{
		"access_token":"eyJAfjskfIWERJFSsfaA…",
		"refresh_token":" eyEHeridzhHkRUPJjdheuvoidf…",
		"token_type":"JWT",
		"expires_in":10000,
		……"shared_key":"AghDF89Wr9Wfwer9"
		}

4	Description	Le client effectue une requête auprès de la ressource protégée en y incluant le jeton d'accès. Au besoin, il encrypte l'information qu'il souhaite envoyer à l'aide de la clé partagée. Ce processus fera l'objet d'une description détaillée dans les sous chapitres 4.2.5.6 et 4.2.7.
	Exemple	**GET /weather** Authorihzation : Bearer eyJAfjskfIWERJFSsfaA…

4.2.5.6 Cas défavorables

Dans chacun des tableaux représentant le flow des processus de récupération du jeton d'accès, seuls les cas favorables où tout se passe bien sont décrits. Le « framework » Oauth2.0 standardise aussi les réponses erronées. Le serveur d'autorisation doit authentifier le client et valider divers paramètres de requête en fonction du processus d'obtention de l'autorisation. En cas d'erreur, il doit en informer le client conformément à la section 5.2 de la RFC6749.

Le serveur d'autorisation répond avec un code http 400 (Bad Request) en incluant le ou les types d'erreur suivants dans le corps de la réponse :

- Invalid_request : un paramètre dans la requête est invalide ou manquant
- Invalid_client : le client n'a pas pu être authentifié
- Invalid_grant : le code d'autorisation ou le jeton de rafraichissement est expiré ou révoqué.
- Unauthorized_client : le client n'est pas autorisé à utiliser cette méthode d'obtention du jeton
- Unsupported_grant_type : la méthode d'obtention de l'autorisation n'est pas supportée par le serveur d'autorisation

Par ailleurs, les flux d'obtentions du jeton d'accès « implicit » et « authorization code » prévoient des exceptions. En effet, ces deux processus impliquent que l'utilisateur de l'application tierce soit automatiquement redirigé vers une url. Dans certain cas, Oauth2.0 préconise de ne pas rediriger le détenteur de la ressource et de l'informer de l'erreur.

Si le serveur d'autorisation n'est pas capable d'authentifier le client ou n'arrive pas à valider l'url de redirection fourni en paramètre (point 3 des flux susmentionné), l'utilisateur doit en être informé sans redirection. Dans tous les autres cas, le serveur d'autorisation informe le client de l'erreur avec les types définis par le standard.

4.2.6 Chiffrement pour les ressources ne pouvant fournir une connexion sécurisée

Certains des IoT choisis pour ce travail de Bachelor ne sont pas suffisamment puissants pour offrir une communication sécurisée avec le protocole TLS. Afin de pallier à ce problème le serveur d'autorisation doit pouvoir émettre des clés de chiffrement à destination des clients et des ressources. Étant donné que le protocole Oauth2.0 est très fortement dépendant du protocole TLS, les éléments suivants ne font pas partie du standard. En outre, afin de simplifier le développement du serveur d'autorisation prototype, seuls les algorithmes de cryptage symétrique sont supportés.

Si la communication entre la ressource protégée et le client ne peut être sécurisée via TLS, le serveur génère une clé de chiffrement symétrique partagée (shared key) qui permettra d'encrypter les messages transitant entre les deux entités. La longueur de la clé dépend du type de cryptage supporté et est déterminé à l'enregistrement de la ressource protégée ciblée. Dans chacun des quatre processus d'obtention d'autorisation, la clé de chiffrement partagée est retournée par le serveur d'autorisation à la même étape que le jeton d'accès. Le processus qui décrit le transfert de données

chiffrées entre le serveur de ressource et le client est l'objet du sous chapitre 4.2.9 « proof of possession ».

Contre-indications :

Étant donné que le flux « implicit » ne prévoit pas de méthode pour remplacer un JWT échu et que l'absence de TLS entre le client et la ressource contraint à mettre en place des systèmes de sécurité très restrictifs (délais d'expiration très court, nonce, etc…), ce flux est contre indiqué dans cette démarche. De plus, la clé de chiffrement est exposée au « user-agent » de la même manière que le jeton d'accès, ce qui pose des risques de sécurité supplémentaires.

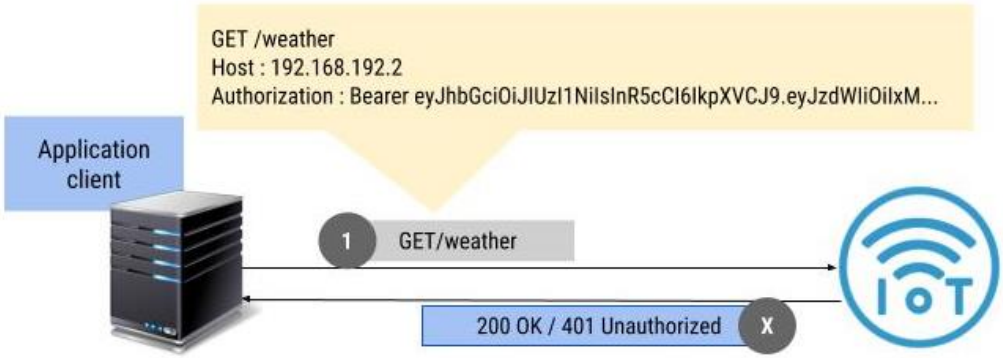
4.2.7 Accéder à une ressource

Afin d'accéder à une ressource protégée, le client doit lui présenter son jeton d'accès. Celui-ci est validé par le serveur de ressource. La méthode utilisée pour vérifier le jeton est en dehors du champ d'application du « framework » Oauth2.0. Notre prototype utilise la spécification supplémentaire « Token Introspeciton » pour réaliser cette tâche et est l'objet du sous chapitre suivant (4.2.8).

En outre, Oauth2.0 ne spécifie pas le type ou la forme des jetons d'accès, ni le protocole à suivre pour le transmettre à une ressource. Le prototype du serveur d'autorisation supporte uniquement les structures JWS et JWE qui font parties des jetons de sécurité nommés « Bearer token ». C'est le standard RFC 6750 qui a la responsabilité de décrire leur usage.

Le jeton d'accès est transmis via une requête http en utilisant l'en-tête « Authorization ». La figure suivante schématise l'interaction entre un client un une ressource protégée.

Figure 16 : Transmission du token



La ressource valide le jeton d'accès en utilisant la méthode de son choix. En cas d'erreur, le serveur de ressource doit en informer le client. Le type et la forme de la réponse est un choix d'implémentation. Pour l'ensemble de nos IoT, une requête valide conduit à une réponse avec un code http 200 tandis qu'une requête invalide conduit à un code http 401 et, suivant le type d'IoT, un message d'erreur.

4.2.8 Token Introspection

L'introspection de jeton dit « Token Introspection » est une spécification supplémentaire du protocole Oauth2.0 standardisé par la RFC 7662. L'objectif de l'introspection de jeton est de fournir à une ressource protégée authentifiée la possibilité de déléguer, tout ou en partie, au serveur d'autorisation la responsabilité de valider un jeton d'accès soumis par un client. En effet, comme nous l'avons vu dans la section traitant des JOSE, certains « token » contiennent une quantité de « metadata » ayant notamment pour objectif de décrire la portée de leur autorisation (expiration, audience, privilège, etc…).

Figure 17 : Token introspection

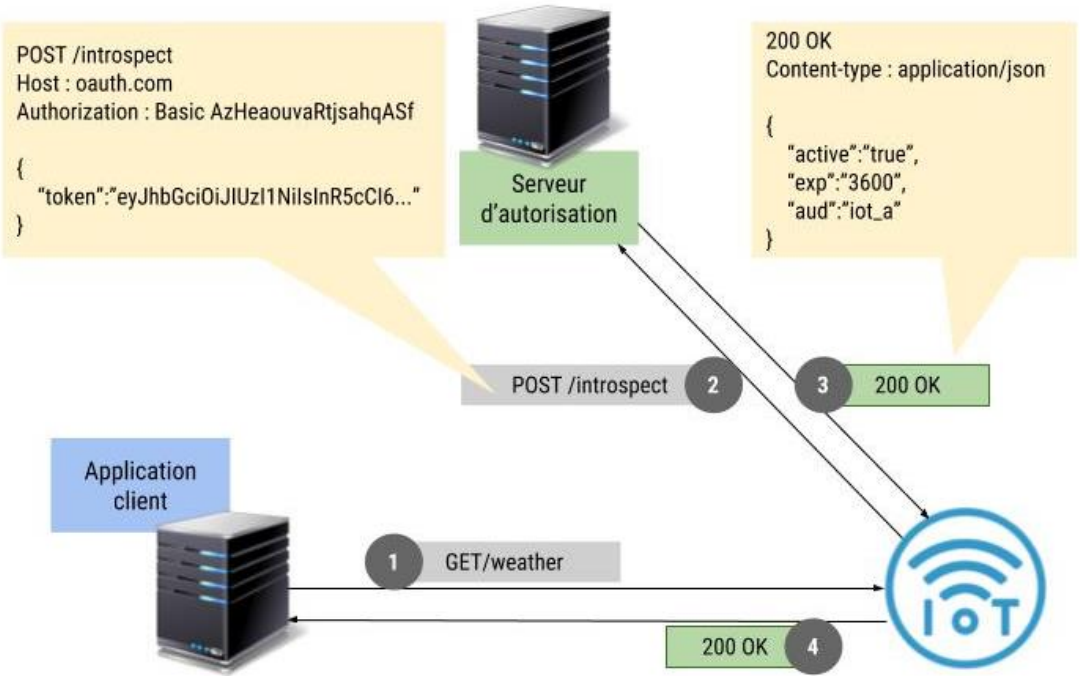


Tableau 12 : Flux de communication token introspection

1	Description	Un client possédant un jeton d'accès déclenche le processus d'introspection de jeton en contactant le service http cible de la ressource protégée. Il inclut le jeton à la requête.
	Exemple	**GET /weather** Authorization : Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp…

2	Description	La ressource protégée peut valider une partie du jeton, par exemple les privilèges requis pour accéder au service. En fonction de cette vérification, elle peut rejeter immédiatement la requête ou transmettre le « token » au serveur d'autorisation via une requête de type POST. La ressource inclue ses identifiants de connexion dans la requête.
	Paramètre	**token :** Requis : le jeton d'accès **token_type_hint :** Optionnel : une information supplémentaire concernant le jeton comme par exemple l'algorithme utilisé pour construire une signature digitale.

	Exemple	**POST /introspect**
		Authorization : Basic AzHeaouvaRtjsahqASf
		Content-type : application/json
		{
		"token ":"eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp…"
		"token_type_hint":"HS256"
		}

3	Description	Le serveur d'autorisation authentifie la ressource et valide le jeton d'accès. Il répond favorablement ou défavorablement après introspection avec un code http 200.
	Paramètre	**active :** Requis : un boolean exprimant l'état de la validité du jeton (true, false) **scope :** Optionnel : les privilèges que contient le jeton **client_id :** Optionnel : l'identifiant unique du client **username :** Optionnel : un identifiant lisible par un humain qui désignant le détenteur de la ressource **token_type :** Optionnel : le type du jeton d'accès En outre, le serveur d'autorisation peut inclure tout autre claim standard appartenant au JWT (exp, iss, aud, etc…).
	Paramètre non-standardisé	**shared_key :** Optionnel : une clé de chiffrement symétrique sans caractères spéciaux partagée entre le client et la ressource protégée qui permettra de crypter les données transmises entre le client et la ressource protégée ou d'établir la preuve de la possession. Ces points sont respectivement traités dans les chapitres 4.2.5.6 et 4.2.9.

	Exemple	**200 OK**
		Content-type : application/json
		{
		"active ":true
		"exp":3600
		}

4	Description	En fonction de la validité du jeton d'accès, la ressource accepte ou rejette la requête avec les codes http 200/401

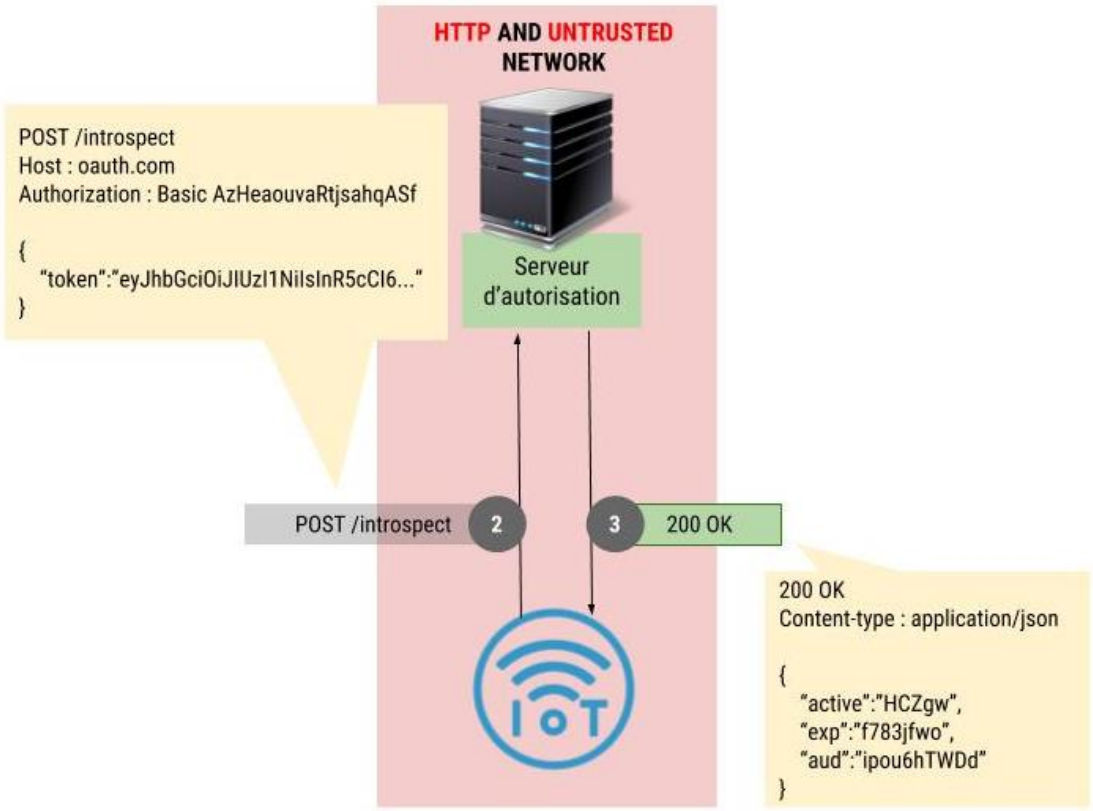
	Exemple	**200 OK**
		Content-type : application/json
		{
		"weather ":"bright sunshiny day"
		}

4.2.8.1 Ressource ne pouvant émettre de requêtes https

Dans le cadre de ce travail de Bachelor, l'Arduino UNO (un des IoT sélectionnés) est incapable de gérer le protocole TLS que ce soit en tant que client ou en tant que serveur. Sans support du https, notre ressource protégée est sensible à l'attaque de type « man-in-the-middle ». Il s'agit d'une attaque couramment utilisée sur des réseaux où les communications ne sont pas cryptées. Elle vise à intercepter les communications entre deux parties.

Il est possible de partiellement résoudre ce problème de sécurité en encryptant les données directement au niveau de la couche message. Lorsque le développeur ou tout autre utilisateur final enregistre la ressource protégée au près du serveur d'autorisation, il négocie les modalités de cryptage souhaité. Si le serveur de ressource est enregistré comme ne pouvant contacter le serveur d'autorisation sous TLS, celui-ci utilisera le « resource_secret » généré automatiquement à l'enregistrement pour encrypter les informations de la requête. Le prototype du serveur d'autorisation supporte uniquement l'algorithme de cryptage symétrique AES en mode ECB et CBC.

Figure 18 : RS – AS sans HTTPS



Il est important de préciser que ce système fonctionne uniquement en partant du principe que la ressource protégée est capable de maintenir la confidentialité de son mot de passe faisant office de clé de cryptage (resource_secret). De ce fait, elle présentera son identifiant unique sans le mot de passe au serveur d'autorisation pour éviter que celui-ci ne puisse être intercepté et que les communications ne soient compromises. En outre, ce dernier point signifie que le serveur d'autorisation ne peut pas authentifier la ressource protégée, ce qui déroge au protocole prévu pour l'introspection de jeton. Dans l'état actuel, le prototype n'implémente pas de méthode supplémentaire permettant de clairement identifier et authentifier un serveur de ressource sans connexion sécurisée. Par conséquent, il est possible d'usurper l'identité d'une ressource et d'introspecter un jeton d'accès sans en avoir l'autorisation. La réponse restera tout de même ininterprétable sans la clé de cryptage. En outre, dans le cadre d'une attaque par le milieu et par rejeu, il est possible pour un attaquant de copier une réponse d'introspection et de se faire passer pour le serveur d'autorisation dans le but de faire croire à une ressource qu'un jeton d'accès est valide. Pour corriger cette vulnérabilité (attaque par le rejeu), la ressource protégée doit pouvoir conserver un identifiant unique (nonce)

retourné par le serveur d'autorisation et considérer toute réponse identique comme étant compromise.

4.2.8.2 Risque lié au choix du mode de cryptage

Comme précisé plus haut, le prototype du serveur d'autorisation supporte uniquement l'algorithme de cryptage symétrique AES (128, 192, 256) en mode ECB et CBC.

Il existe un risque notable en utilisant le mode ECB. En effet, il n'utilise pas de vecteur d'initialisation (IV) ce qui signifie que deux messages identiques le seront aussi une fois encryptés. Pour rappel, le serveur d'autorisation à l'obligation de répondre à une requête d'introspection avec au moins le paramètre « active » et comme valeur « true » ou « false ». Avec le mode ECB, chaque message encrypté avec le paramètre « active » à « true » sera identique et donc potentiellement réutilisable par un attaquant.

Le mode CBC permet d'éliminer le risque puisqu'il garantit que deux messages identiques seront sensiblement différents une fois encryptés. A noter que le serveur d'autorisation devra transmettre le vecteur d'initialisation dans la réponse d'introspection, sans quoi la ressource ne pourra décrypter le message.

4.2.9 Proof of Possession (PoP)

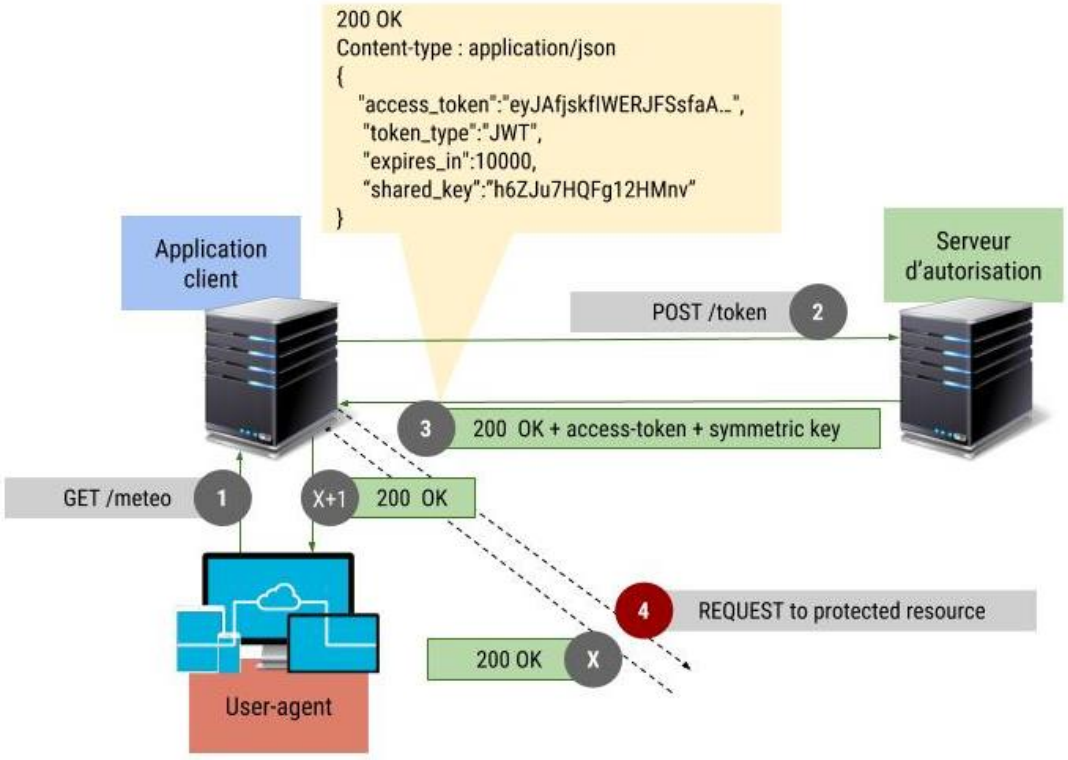
La spécification Oauth2.0 sur les « Bearer Token » standardisée par la RFC 6750 défini que tout parti en possession d'un jeton d'accès de ce type à le droit d'accéder à la ressource protégée associée sans démontrer qu'il a le droit de l'utiliser (ou de le posséder) Cela signifie qu'un serveur de ressource n'a pas de garantie qu'un jeton d'accès soumis par un client, ait réellement été émis pour celui-ci. Par analogie, le traditionnel nom d'utilisateur et mot de passe qu'on utilise tous les jours pour s'authentifier pose exactement le même problème. En effet, il y a une différence entre authentifier et identifier. Un mot de passe permet d'authentifier un utilisateur mais pas de l'identifier. Il n'offre pas de garantie que la personne qui l'utilise est bien le propriétaire de la ressource à laquelle il donne accès. Pour réduire ce risque, certains mécanismes peuvent être mis en place comme par exemple le « Two-factor authentication (2FA) ». Cette méthode permet d'identifier un utilisateur en lui demandant de transmettre en plus de son mot de passe, un code d'authentification qui lui a été préalablement envoyé par sms ou email. Par conséquent, le mot de passe à lui seul, ne permet plus d'accéder à une ressource et c'est exactement l'objectif de la spécification supplémentaire Proof-of-Possession : éviter qu'un jeton d'accès à lui seul ne suffise à accorder l'accès à une ressource protégée. Il existe à ce jour pas de standard décrivant l'architecture du PoP mais l'IETF a publié une version préliminaire sur ce sujet à titre d'information.

Le processus qui permet d'obtenir la preuve de la possession diffère sensiblement en fonction du type de clé utilisé (symétrique ou asymétrique). Étant donné que le prototype du serveur d'autorisation ne supporte que les clés symétriques, seul ce flux sera présenté dans le cadre de ce travail de Bachelor.

4.2.9.1 Obtention de la clé symétrique

Dans le sous chapitre 4.2.5.6 traitant de la clé de chiffrement partagée, nous parlons brièvement de la façon dont on peut encrypter les données transitant entre un client et une ressource protégée s'il est impossible de communiquer sous TLS. Cette clé de chiffrement symétrique n'est rien d'autre que la clé qui va nous permettre d'effectuer la preuve de la possession. Comme indiqué dans ce chapitre, la clé est délivrée en même temps que le jeton d'accès dans chacun des processus d'obtention de l'autorisation.

Figure 19 : Proof-of-possession – acquisition de la clé symétrique



La Figure 19 modélise l'obtention d'une clé symétrique par un client au moyen du processus « client credentals ».

4.2.9.2 Interaction entre le serveur de ressource et le client avec TLS

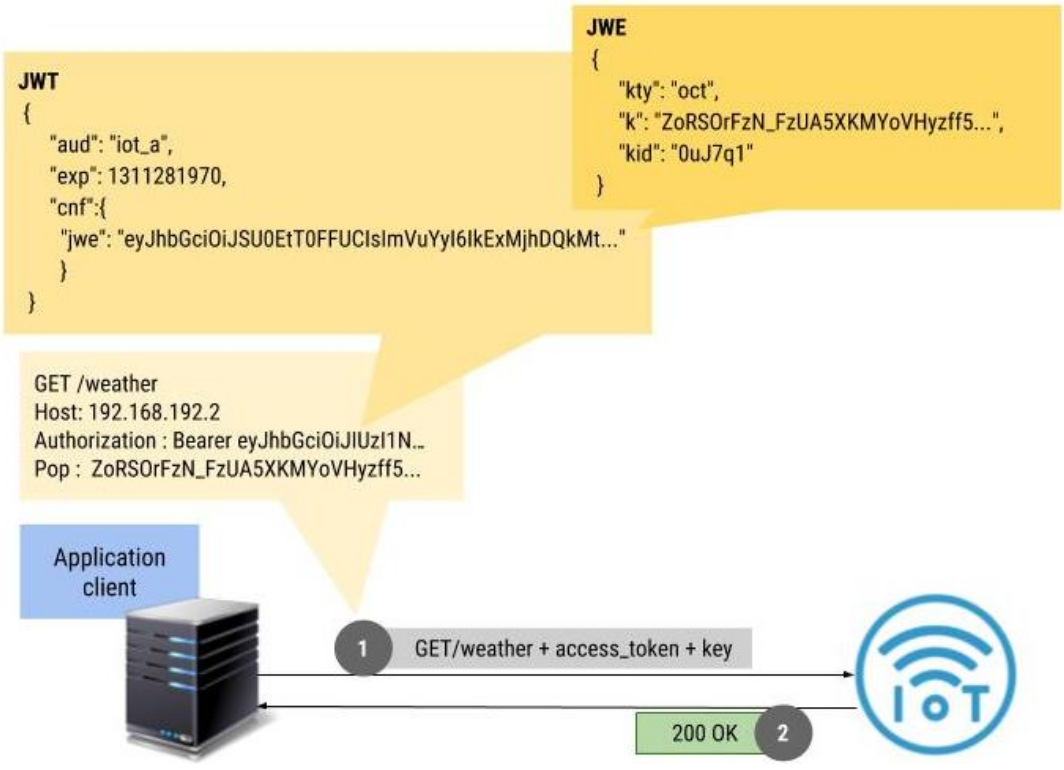
Afin de prouver au serveur de ressource que le client est autorisé à utiliser un jeton d'accès, celui-ci doit inclure dans la requête adressée à la ressource, la clé symétrique et le dudit jeton. La ressource à ensuite la responsabilité de vérifier que la clé est

identique à celle qui a été émise par le serveur d'autorisation. Il existe deux méthodes permettant au serveur de ressource d'obtenir de façon sécurisée la clé créée par le serveur d'autorisation : via le jeton d'accès ou via l'introspection de jeton (Token Introspection). L'utilisation de l'un ou l'autre de ces processus est un choix d'implémentation qui dépend fortement de la capacité de la ressource à effectuer des opérations sur les JWT.

Solution 1 : clé symétrique obtenu via un JWE :

Comme vu dans le chapitre traitant de la famille des JOSE, le standard JWE offre une structure pour créer des JWT dont le « payload » est encrypté. Grâce à ce type de jeton de sécurité, il est possible de transporter de manière opaque de l'information. Cette méthode d'interaction avec le serveur de ressource propose d'empaqueter un JWE à l'intérieur d'un JWT. La RFC 7800 standardise la sémantique à utiliser avec un JWT pour présenter une clé permettant d'effectuer la preuve de la possession.

Figure 20 : PoP – JWK/JWE/JWT



Quand le serveur d'autorisation émet le jeton (JWT) d'accès et la clé de cryptographie symétrique (JWK) permettant d'effectuer la preuve de la possession, il encrypte la JWK à l'intérieur d'un JWE en utilisant le secret « ressource_secret » de la ressource protégée cible (créé à l'enregistrement du serveur de ressource). De ce fait, seul le

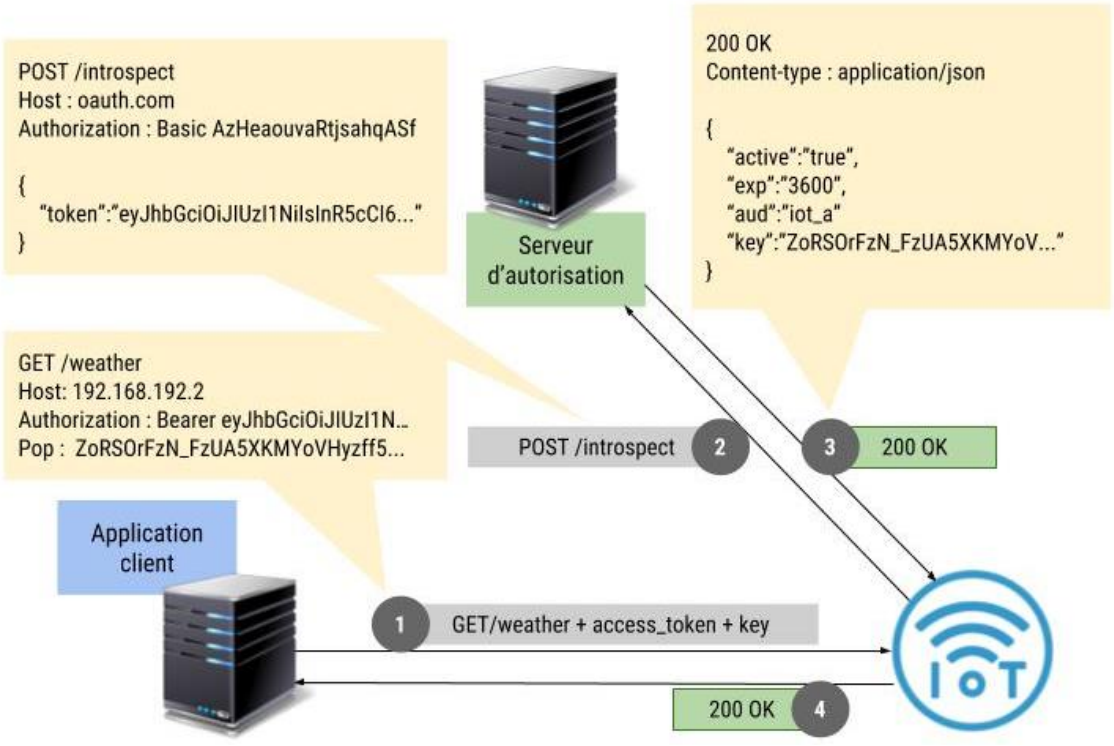
receveur du jeton d'accès est en mesure de lire la clé symétrique. Le client effectue une requête pour accéder à la ressource en suivant le protocole pour les jetons de type « Bearer Token » et inclue en plus la clé symétrique obtenue précédemment. A noter, que dans le schéma ci-dessus, l'en-tête « pop » est utilisée pour transmettre la JWK. Je n'ai trouvé aucune indication ou bonne pratique sur comment former une requête avec une clé de cryptographie pour faire du Proof-of-possession.

La ressource protégée à ensuite la responsabilité de comparer les deux clés obtenues.

Solution 2 : clé symétrique obtenu via l'introspection de jeton

La seconde solution pour obtenir la clé symétrique permettant de faire la preuve de la possession tire parti de la spécification supplémentaire « Token Introspection » vu au sous chapitre 4.2.8. Pour rappel, ce processus a pour but de déléguer au serveur d'autorisation la responsabilité de valider le jeton d'accès. Étant donné que celui-ci est en mesure d'inclure dans sa réponse un certain nombre de paramètres, il peut tout à fait y inclure la clé symétrique.

Figure 21 : Pop - Token Introspection



Quand le serveur d'autorisation émet le jeton (JWT) d'accès et la clé de cryptographie symétrique (JWK) permettant d'effectuer la preuve de la possession, il stock la JWK dans l'attente d'une requête d'introspection. Le client effectue la même requête que dans

la méthode précédente. La ressource protégée valide le jeton auprès du serveur d'autorisation qui lui retourne en plus de la réponse standard, la clé de cryptographie symétrique. A noter que cette clé est incluse dans la réponse si, et seulement si, le jeton d'accès est considéré comme valide.

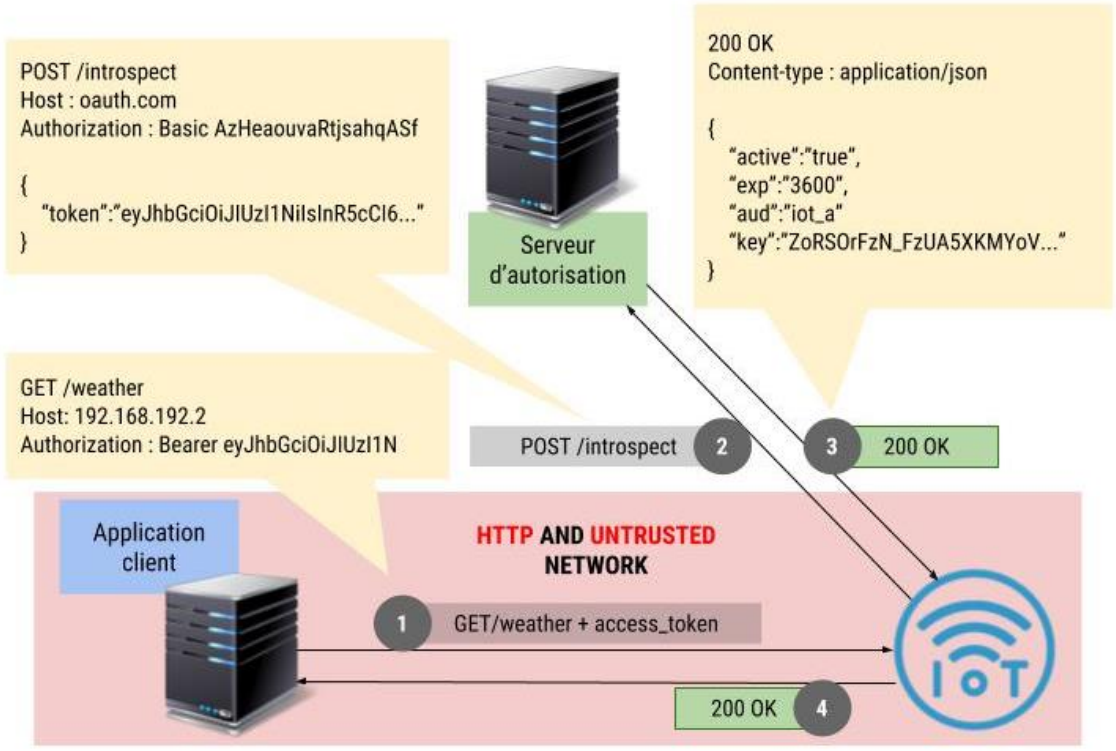
Comme dans le point précédent, la ressource protégée compare les deux clés.

4.2.9.3 Interaction entre le serveur de ressource et le client sans TLS

Afin d'utiliser le protocole Oauth2.0 pour sécuriser un réseau d'IoT, il est nécessaire de prendre en compte certains facteurs limitants propre à ce type de ressource et de trouver des solutions. Les principaux facteurs limitants que j'ai rencontrés sont principalement liés au manque de mémoire et de puissance de certains IoT. Quand on travaille avec des JWT et Oauth2.0 en général, il faut s'attendre à devoir gérer des requêtes transportant de longues chaines de caractères et cela peut être problématique sur certains IoT comme l'Arduino UNO. La solution consiste à transporter le strict minimum d'informations requises permettant d'atteindre les objectifs de sécurité fixés. En second lieu, il existe une autre famille de jetons d'accès nommé « CBOR Object Signing and Encryption (COSE) » (RFC 8152) standardisé par le « Concise Binary Object Representation (CBOR) » (RFC 7049) qui offre une structure pour créer des « CBOR Web Token (CWT) ». L'objectif des CWT est d'offrir le même système de représentation d'une autorisation qu'un JWT mais optimisé pour des appareils ayant des capacités très limitées. Malheureusement, je n'ai trouvé aucune librairie activement maintenue pour ces types de jetons d'accès.

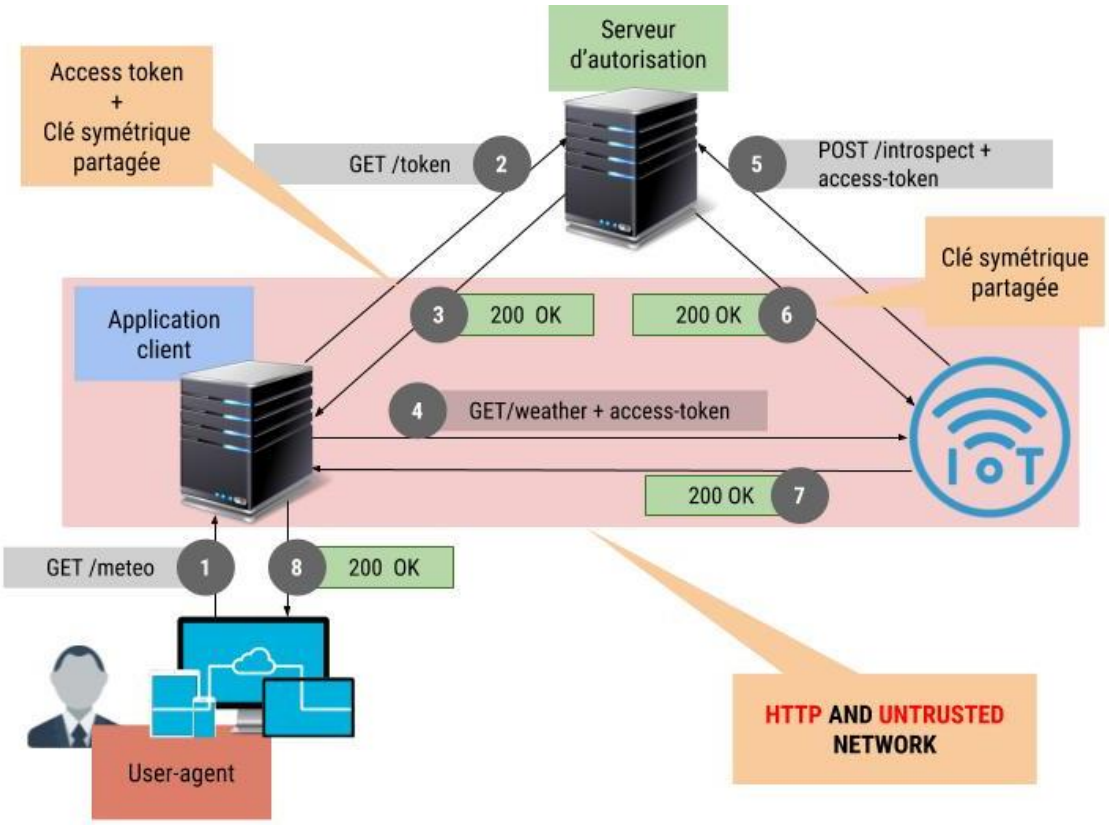
En outre, certains IoT ne sont pas assez puissants pour communiquer avec le protocole TLS. Si l'interaction entre le client et la ressource protégée ne peut être cryptée au niveau de la couche réseau (https), le client ne peut pas transmettre la clé symétrique de façon traditionnelle. En effet, l'absence de TLS expose la ressource protégée à une attaque de type « man in the middle ». Jacob Ideskog a proposé une solution basée sur la preuve de la possession lors d'une conférence organisée par Nordic APIS. Comme mentionné dans le chapitre sur les clés de chiffrement partagées, l'idée est que le serveur d'autorisation puisse émettre au même titre que pour le « proof-of-possession » une clé symétrique qui sera partagée entre le client et la ressource cible. Seulement au lieu de comparer les deux clés pour obtenir la preuve que le client est autorisé à utiliser un jeton d'accès, les deux entités utilisent cette clé cryptographique partagée pour directement encrypter les données qu'ils se transmettent.

Figure 22 : PoP – Token Introspection sans TLS



Dans la Figure 22, le client obtient la clé symétrique de façon standard comme décrit précédemment. Puis, il contacte la ressource protégée en lui adressant le jeton d'accès. Le serveur de ressource valide le jeton auprès du serveur d'autorisation et récupère la clé symétrique. Ensuite, il encrypte les données et les retourne au client qui peut à son tour utiliser la clé partagée pour décrypter les données. Ce processus peut aussi être mis en œuvre en utilisant la méthode d'obtention de la clé symétrique via un JWE. Contrairement au concept initial de la preuve de la possession proposé par Oauth2.0, la ressource cible n'est pas en mesure de déterminer si le client a l'autorisation de posséder ou d'utiliser le jeton d'accès. Quoi qu'il arrive, si le jeton est considéré comme valide, la ressource protégée formera une réponse encryptée. Cependant, si un attaquant tante de récupérer des données à partir d'un jeton d'accès volé, il ne pourra pas les décrypter. Un JWT ne permet donc plus à lui seul de récupérer des données directement exploitables et rejoint donc notre objectif initial.

Figure 23 : Client Credentials - PoP – Token Introspection sans TLS



La Figure 23 schématise l'obtention d'un jeton d'accès et de la clé partagée par le client avec la méthode « client credentials » ainsi que l'intropsection de token permetant à la ressource de valider le « token » et d'obtenir la clé partagée faisant office de preuve de possession.

Il est aussi possible de tirer certains avantages des JWT pour sécuriser notre processus de communciation sans TLS. Les jetons d'accès doivent obligatoirement contenir les claims suivant :

- Exp : paramétrer un délai d'expiration très court (de l'ordre de quelque seconde)
- Jti : un identifiant unique. Le serveur d'autorisation garde en mémoire cache l'identifiant unique du JWT jusqu'à son expiration. Il rejette tout jeton d'accès ayant un identifiant déjà utilisé.
- Aud : l'identifiant unique de la ressource cible

Ces mesures ont pour objectif de limiter le risque de subir une attaque par rejeu. C'est particulièrement important de se prémunir de ce type d'attaque quand le client à la possibilité de modifier l'état d'un IoT ou de faire des opérations CRUD sur les données.

En outre et comme pour le chapitre consacré à l'introspection de jeton, il est nécessaire d'encrypter les données avec un algorithme qui permet l'emploi de vecteurs d'initialisation afin de réduire, une fois encore, le risque d'attaque par rejeu.

5. Conclusion

En effectuant ce travail de Bachelor, j'ai pu expérimenter le protocole Oauth2.0 et ses variantes appliquées aux IoT. J'ai pu constater quels étaient les difficultés de mise en place et me suis fait une bonne idée de ce qui peut fonctionner dans le cadre d'une utilisation réel et de ce qu'il ne faut pas faire.

Oauth2.0 est un protocole qui offre de solides garanties à une ressource protégée qui lui délègue son processus d'autorisation. Les IoT sont aussi des ressources et finalement les différences d'implémentation résident principalement dans l'absence de TLS pour certains IoT.

Dans ce travail, deux problématiques propres aux IoT sont étudiées :

Problématique 1 : pas de TLS entre le client, la ressource et le serveur d'autorisation :

L'Arduino UNO a posé un gros défi d'implémentation à cause de la quantité de mémoire disponible limitée. A de multiples reprises, la SRAM à fait défaut causant sans doute des fuites de mémoire et des débordements de pile. Avec 98 bytes de mémoire restante à l'exécution de la requête permettant d'obtenir des données de la ressource protégée, la stabilité du prototype mis en place est très difficile à garantir.

Le manque de mémoire m'oblige aussi à utiliser le mode d'opération ECB de l'algorithme AES. Ce mode n'utilise pas de vecteur d'initialisation ce qui signifie que si l'on encrypte deux messages strictement identiques, ils le seront aussi une fois encryptés ce qui permet d'effectuer un profil statistique en fonction du block. Au contraire le mode CBC avec un vecteur d'initialisation permet de garantir que deux messages identiques ne le seront plus une fois encryptés. Ce mode requiert un peu plus de mémoire à allouer à cause du vecteur d'initialisation et déclenche dans l'état actuel un débordement de mémoire tampon qui se traduit par le redémarrage de l'appareil. Toutefois, il est possible de contourner cette vulnérabilité en forçant le client à systématiquement soumettre une nouvelle clé de chiffrement partagée auquel cas, un même message sera systématiquement encrypté avec une clé différente et par conséquent différent une fois encrypté.

Mais le principal problème vient de l'absence de TLS entre la ressource et le serveur d'autorisation lors de l'introspection de jeton d'accès. En effet, comme expliqué à la fin du sous chapitre 4.2.8.1 sur le « token introspection » sans TLS, un attaquant peut usurper l'identité du serveur d'autorisation et faire croire à notre ressource protégée que le jeton d'accès soumis est valide. Il est possible de redresser cette vulnérabilité en conservant un identifiant unique de chaque réponse du serveur d'autorisation mais c'est impossible à implémenter pour l'Arduino UNO, qui souffre déjà d'un espace mémoire très limité. La seconde solution est de transmettre la clé partagée via un JWE dans un JWT (PoP 4.2.9.1, solution 1) et de laisser la responsabilité de la validation à l'IoT. A nouveau, cette solution est peu envisageable pour l'Arduino UNO car le JWT serait beaucoup trop long et poserait des problèmes de mémoire.

En conclusion à la vue des points mentionnés ci-dessus et des alternatives viables qui commencent à émerger sur le marché, je suis d'avis de dire que pouvoir effectuer des requêtes https au près d'un serveur d'autorisation est le prérequis minimum pour sécuriser un IoT.

Problématique 2 : pas de TLS entre le client et la ressource :

Dans le sous chapitre 4.2.9.3 traitant des interactions entre le client et la ressource protégée sans TLS, nous avons vu qu'il est tout à fait possible, moyennant certaines modifications du protocole Oauth2.0, d'établir une connexion sécurisée et encryptée entre les deux entités. Cette solution fonctionne bien mais implique qu'il faut utiliser des JWT avec des délais d'expiration très courts et non-réutilisables ce qui force un client à demander un nouveau jeton d'accès pour chaque requête qu'il veut adresser à la ressource. Ce dernier point pose un problème dans le cadre de l'utilisation du flux d'obtention de l'autorisation « implicit » qui ne permet pas d'obtenir de jeton de rafraichissement. Le détenteur de la ressource devrait recommencer le processus d'authentification au près du serveur d'autorisation à chaque requête. En outre, avec cette contrainte au niveau des « token », le serveur d'autorisation peut être énormément sollicité et il faut donc implémenter une solution scalable. De ce fait, il peut être judicieux de choisir un langage et une architecture adaptés à ce genre de défi.

Oauth2.0 : une solution parmi d'autre :

Le protocole Oauth2.0 est un standard relativement lourd à implémenter et qui sera particulièrement attrayant dans une dimension industrielle. En effet, il offre des solutions pour tout type d'application et bien qu'il ne soit pas prévu pour des IoT à la base, est

facilement adaptable. Ce protocole est notamment très utilisé avec les micro services et répond parfaitement au besoin des systèmes hautement distribués.

Cependant, les objets connectés ne sont pas seulement utilisés dans un contexte industriel. Ce sont aussi des outils personnels de la vie quotidienne. Oauth2.0 est une solution trop compliquée et trop lourde à utiliser pour une personne n'ayant pas des connaissances dans le domaine de l'informatique (en l'état actuelle). En outre, les besoins en sécurité diffèrent en fonction du cadre d'utilisation. En effet, une station météo personnelle (domestique) n'aura pas besoin des mêmes garanties de sécurité qu'un réseau de caméras de surveillance. Le célèbre antivirus Bitdefender propose par exemple, un dispositif permettant de se substituer au routeur et de centraliser les communications des différents IoT en un même point. Il permet aussi de gérer les appareils connectés au réseau. Cependant, ces appareils sont enregistrés automatiquement auprès de la « Bitdefender box » via un scan complet du réseau (adresse MAC et IP). La sécurité est donc surtout garantie par l'intégrité du réseau et n'est pas du même niveau que Oauth2.0.

Finalement, ce qu'il manque encore aux IoT, c'est une stratégie cohérente de sécurité, un standard vers lequel se diriger qui permettrait à monsieur tout le monde d'acheter un objet connecté, de l'enregistrer avec une procédure simplifiée auprès d'un serveur d'autorisation, et d'utiliser une application pour accéder aux données de son bien nouvellement acquis depuis n'importe où dans le monde.

6. Glossaire

AES (Advanced Encryption Standard) : est un algorithme de cryptographie symétrique.

AES MODE ECB (Electronic Code Book) : est un mode d'opération de cryptographie qui consiste à subdiviser un message en plusieurs blocs qui sont chiffrés séparément. Deux messages identiques seront chiffrés de la même manière.

AES MODE CBC (Cipher Block Chaining) : est un mode d'opération de cryptographie qui utilise un vecteur d'initialisation (IV) permettant de rendre chaque message unique.

CALLBACK : fonction de rappel qui est passé en paramètre d'une autre fonction.

MAN-IN-THE-MIDDLE : attaque qui a pour objectif de se placer entre deux parties qui communiquent sans cryptographie et d'en intercepter les communications.

REPLAY-ATTACK : attaque qui utilise une transmission interceptée et qui tente de la répéter.

JSON (Javascript Object Notation) : un format de données textuel léger.

FORK : un logiciel créé à partir du code source d'un logiciel existant.

IDE : un environnement de développement intégré.

NONCE : un nombre aléatoire destiné à être utilisé qu'une seule fois. Il permet de prévenir les attaques par rejeu.

SHA-1 : une fonction de hachage cryptographique qui produit un résultat de 160 bits.

MAC (Message Authentication Code) : un code accompagnant des données dans le but d'en assurer l'intégrité.

TIMESTAMP : représente le nombre de secondes qui s'est écoulé depuis le 1er janvier 1970.

CRYPTOGRAPHIE SYMETRIQUE : permet de chiffrer et déchiffrer des messages à l'aide d'un même mot de passe.

Bibliographie

Gartner - Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016 [en ligne] 7 février 2017 [Consulté le 20 juin 2018] Disponible à l'adresse : https://www.gartner.com/newsroom/id/3598917

Wikipédia – Gartner [en ligne] [Consulté le 20 juin 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Gartner

Janua - La sécurité de l'IoT – Internet des Objets [en ligne] [Consulté le 20 juin 2018] Disponible à l'adresse : http://www.janua.fr/la-securite-de-liot-internet-des-objets/

Wikipédia – Machine To Machine [en ligne] [Consulté le 20 juin 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Machine_to_machine

Quora - In OAuth 2.0, how do resource servers assert a token issued by an authorization server [en ligne] [Consulté le 29 juin 2018] Disponible à l'adresse : https://www.quora.com/In-OAuth-2-0-how-do-resource-servers-assert-a-token-issued-by-an-authorization-server

JONES et al, 2015, JSON Web Signature (JWS) [en ligne] [Consulté le 29 juin 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7515

JONES et al, 2015, JSON Web Encryption (JWE) [en ligne] [Consulté le 29 juin 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7516

JONES et al, 2015, JSON Web Token (JWT) [en ligne] [Consulté le 29 juin 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7519

HARDT, MICROSOFT, 2012, The OAuth 2.0 Authorization Framework [en ligne] [Consulté le 29 juin 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc6749

RICHER, 2015, OAuth 2.0 Token Introspection [en ligne] [Consulté le 29 juin 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7662

Youtube - OAuth 2.0 and the Internet of Things (IoT) [vidéo] [Consulté le 30 juin 2018] Disponible à l'adresse : https://www.youtube.com/watch?v=ZF0wrHtiXYw&list=FLDLbsosBjJyowp_MXmf7Tdg

Klostercluster - Convert a certificate from PEM to hex for embedding into C-code [en ligne] 25 janvier 2017 [Consulté le 15 juillet 2018] Disponible à l'adresse : https://klostercluster.dkrz.de/wordpress/m214089/2017/01/25/convert-a-certificate-from-pem-to-hex-for-embedding-into-c-code/

GithubGist – docker install on Raspberry Pi [en ligne] [Consulté le 24 juillet 2018] Disponible à l'adresse : https://gist.github.com/tyrell/2963c6b121f79096ee0008f5a47cf347

FreeCodeCamp - The easy way to set up Docker on a Raspberry Pi [en ligne] 28 mai 2018 [Consulté le 24 juillet 2018] Disponible à l'adresse : https://medium.freecodecamp.org/the-easy-way-to-set-up-docker-on-a-raspberry-pi-7d24ced073ef

Github - RestClient for Arduino [en ligne] [Consulté le 30 juillet 2018] Disponible à l'adresse : https://github.com/csquared/arduino-restclient

Github - phpseclib - PHP Secure Communications Library [en ligne] [Consulté le 5 août 2018] Disponible à l'adresse : https://github.com/phpseclib/phpseclib

VALSORDA, Filippo, 2013, THE ECB PENGUIN [en ligne] 10 novembre 2013 [Consulté le 5 août 2018] Disponible à l'adresse : https://blog.filippo.io/the-ecb-penguin/

Stackexchange - Why shouldn't I use ECB encryption PENGUIN [en ligne] [Consulté le 5 août 2018] Disponible à l'adresse : https://crypto.stackexchange.com/questions/20941/why-shouldnt-i-use-ecb-encryption

Wikipédia - Transport Layer Security PENGUIN [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Transport_Layer_Security

Wikipédia - Arduino Uno [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/Arduino_Uno

Wikipédia - Microchip Technology [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/Microchip_Technology

Wikipédia - ATmega328 [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/ATmega328

Wikipédia - Flash memory [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/Flash_memory

Wikipédia - Static random-access memory [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/Static_random-access_memory

Wikipédia - EEPROM [en ligne] [Consulté le 6 août 2018] Disponible à l'adresse : https://en.wikipedia.org/wiki/EEPROM

RICHER et al, 2016, OAuth 2.0 Proof-of-Possession (PoP) Security Architecture draft-ietf-oauth-pop-architecture-08 [en ligne] [Consulté le 29 août 2018] Disponible à l'adresse : https://tools.ietf.org/html/draft-ietf-oauth-pop-architecture-08

Kantara - OAuth 2.0 Resource Registration [en ligne] 12 mars 2017 [Consulté le 1 septembre 2018] Disponible à l'adresse : https://docs.kantarainitiative.org/uma/wg/oauth-resource-reg-2.0-07.html

Wikipédia - Request for comments [en ligne] [Consulté le 11 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Request_for_comments

Wikipédia - Internet Engineering Task Force [en ligne] [Consulté le 11 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Internet_Engineering_Task_Force

Wikipédia – User agent [en ligne] [Consulté le 12 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/User_agent

FRANKS et al, 1999, HTTP Authentication: Basic and Digest Access Authentication [en ligne] [Consulté le 17 septembre 2018] Disponible à l'adresse : https://tools.ietf.org/html/rfc2617

JONES et al, 2012, The OAuth 2.0 Authorization Framework: Bearer Token Usage [en ligne] [Consulté le 18 septembre 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc6750

Wikipédia – Attaque de l'homme du milieu [en ligne] [Consulté le 18 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Attaque_de_l%27homme_du_milieu

Wikipédia – Double authentification [en ligne] [Consulté le 18 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Double_authentification

JONES et al, 2016, Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) [en ligne] [Consulté le 18 septembre 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7800

JONES, MICROSOFT, 2015, JSON Web Key (JWK) [en ligne] [Consulté le 18 septembre 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7517

SCHAAD, AUGUST CELLARS, 2017, CBOR Object Signing and Encryption (COSE) [en ligne] [Consulté le 18 septembre 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc8152

BORMANN et al, 2017, Concise Binary Object Representation (CBOR) [en ligne] [Consulté le 18 septembre 2018] ISSN 2070-1721 Disponible à l'adresse : https://tools.ietf.org/html/rfc7049

WAHLSTROEM et al, 2015, CBOR Web Token (CWT) draft-wahlstroem-oauth-cbor-web-token-00 [en ligne] [Consulté le 18 septembre 2018] Disponible à l'adresse : https://tools.ietf.org/html/draft-wahlstroem-oauth-cbor-web-token-00

Wikipédia – NodeMCU [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/NodeMCU

Arduino - Arduino Ethernet Shield V1 [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1

Wikipédia – Raspberry Pi [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Raspberry_Pi

Raspberrypi - RASPBERRY PI 3 MODEL B+ [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

Wikipédia – Internet des objets [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Internet_des_objets

Wikipédia – Domotique [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Domotique

Wikipédia – Quantified self [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Quantified_self

Wikipédia – Docker (logiciel) [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Docker_(logiciel)

OSBORNE, Charlie, 2018, ZDNet - 1,5 milliard de dollars pour sécuriser l'IoT en 2018 [en ligne] 22 mars 2018 [Consulté le 20 septembre 2018] Disponible à l'adresse : https://www.zdnet.fr/actualites/15-milliard-de-dollars-pour-securiser-l-iot-en-2018-39865874.htm

JUVIGNY, Julia, 2017, DigitalSecurity - BILAN DES ATTAQUES IOT EN 2016 2018 [en ligne] 16 janvier 2017 [Consulté le 20 septembre 2018] Disponible à l'adresse : https://www.digital.security/fr/blog/bilan-des-attaques-iot-en-2016

Wikipédia – Mode d'opération (cryptographie) [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_(cryptographie)

Wikipédia – Cryptographie symétrique [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Cryptographie_sym%C3%A9trique

Wikipédia – Fonction de rappel [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Fonction_de_rappel

Wikipédia – Attaque par rejeu [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Attaque_par_rejeu

Wikipédia – JavaScript Object Notation [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

Wikipédia – Nonce (cryptographie) [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Nonce_(cryptographie)

Wikipédia – SHA-1 [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/SHA-1

Wikipédia – Code d'authentification de message [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://fr.wikipedia.org/wiki/Code_d%27authentification_de_message

SANDOVAL, Kristopher, 2017, NORDICAPIS - OAuth 2.0 – Why It's Vital to IoT Security [en ligne] 23 mars 2017 Mis à jour le 13 décembre 2017 [Consulté le 20 septembre 2018] Disponible à l'adresse : https://nordicapis.com/why-oauth-2-0-is-vital-to-iot-security/

Bitdefender - Bitdefender BOX [en ligne] [Consulté le 20 septembre 2018] Disponible à l'adresse : https://www.bitdefender.com/box/faq/

Annexe 1 : Liste des dépendances

Librairies crées ou modifiées pour les besoins de ce travail de Bachelor :

- https://github.com/tigerwill90/NodeMCU-Oauth : le programme à exécuter sur le NodeMCU.
- https://github.com/tigerwill90/ArduinoUno-Oauth : le programme à exécuter sur l'Arduino UNO.
- https://github.com/tigerwill90/Raspberrypi-Oauth : l'application à exécuter sur le Raspberry PI.
- https://github.com/tigerwill90/Oauth-IoT : le serveur d'autorisation Oauth2.0
- https://github.com/tigerwill90/esp8266-restclient : un fork d'un client REST pour le NodeMCU. Le nom de la librairie est modifier afin d'éviter des problèmes de collision de nom.
- https://github.com/tigerwill90/RestServer : un fork d'un client REST pour l'Arduino UNO avec des modifications apportée pour supporter le protocole Oauth2.0.

Annexe 2 : Installation du serveur d'autorisation

L'installation du serveur Oauth2.0 nécessite les prérequis suivants :

- Un système linux
- L'outil make
- Docker
- Docker-compose

Système linux :

Il est possible de faire fonctionner le serveur d'autorisation sur Windows, Mac et Linux. Cependant, grâce à la technologie de conteneurisation et a des outils d'automatisation, il est possible de faciliter grandement la mise en place de l'environnement de développement.

L'installation automatique devrait fonctionner sur tout système UNIX mais n'a été testée que sur Debian et Linux.

Make :

Sur la plupart des versions de linux, make devrait être installé d'office. Si ce n'est pas le cas il faut installer les paquets suivants :

- Build-essential
- Make

sudo apt install build-essential make

Docker :

L'outil de conteneurisation est disponible sous Linux, Mac os, Windows serveur, professionnel et étudiant. Il existe une version entreprise (payante) et une version communauté.

Téléchargement et procédure d'installation :

- Windows : https://store.docker.com/editions/community/docker-ce-desktop-windows
- Mac : https://store.docker.com/editions/community/docker-ce-desktop-mac
- Linux : https://docs.docker.com/install/linux/docker-ce/ubuntu/

Docker-compose :

L'outil d'orchestration de conteneurs est disponible sous Linux, Mac os et Windows.

Téléchargement et procédure d'installation : https://docs.docker.com/compose/install/

Oauth2.0

Si vous pouvez satisfaire à toutes les dépendances précédemment citées, il est très simple de démarrer un serveur d'autorisation.

1) Téléchargez le serveur d'autorisation (lien en annexe 1)
2) Déplacez-vous à la racine du dossier précédemment téléchargé
3) Renommez le fichier « .env.exemple ». Ouvrez-le et modifiez la variable d'environnement « ALIAS » avec l'adresse IP de votre machine hôte.
4) Exécutez la commande suivante : make install ou sudo make install. La procédure d'installation devrait commencer. Il y a beaucoup de dépendances à télécharger alors l'installation peut prendre un peu de temps.
5) Si tout c'est bien passé, vous devriez pouvoir atteindre « phpmyadmin » sur l'adresse localhost :8080.
6) Profitez-en pour créer la base de données avec le fichier « oauth.sql » se trouvant dans le sous répertoire « db/model/ »

Les services suivants sont disponibles :

- GET / POST /auth
- GET / POST /introspect
- POST /token

Annexe 3 : Installation du Raspberry PI 3b+

L'installation du serveur Oauth2.0 nécessite les prérequis suivants :

- Raspbian
- L'outil make
- Docker
- Docker-compose

Make :

Sur la plupart des versions de linux, make devrait être installé d'office. Si ce n'est pas le cas il faut installer les paquets suivants :

- Build-essential
- Make

sudo apt install build-essential make

Docker et Docker-compose:

Il est possible d'installer facilement docker/docker-compose sur Raspbian en suivant la procédure décrite au lien ci-dessous :

https://medium.freecodecamp.org/the-easy-way-to-set-up-docker-on-a-raspberry-pi-7d24ced073ef

Installation du service météo sur le Raspberry PI 3B+ :

1) Téléchargez le service météo pour le Raspberry PI (lien en annexe 1)
2) Déplacez-vous à la racine du dossier précédemment téléchargé
3) Renommez le fichier « .env.exemple ». Ouvrez-le et modifiez la variable d'environnement « ALIAS » avec l'adresse IP de votre machine hôte
4) Exécutez la commande suivante : make install ou sudo make install. La procédure d'installation devrait commencer

Le service suivant est disponible :

- GET /rain

Annexe 4 : Installation de l'Arduino IDE

L'Arduino UNO et le NodeMCU ont besoin d'un IDE pour compiler et uploader les programmes écrit en C/C++. Ce logiciel est multiplateforme et peut être téléchargé à l'adresse suivante : https://www.arduino.cc/en/main/software

La première chose à effectuer une fois le logiciel installé est d'ajouter un manageur de librairie permettant de récupérer des modules pour le NodeMCU.

Ajout de dépendances externes :

1) Allez dans l'onglet préférence
2) Ajoutez le lien ci-dessous http://arduino.esp8266.com/stable/package_esp8266com_index.json dans la zone de texte « Additional Boards Manager URLs »
3) Validez et fermez

Ajout de librairies :

On peut ajouter des librairies de deux façons avec l'IDE. Directement depuis le « store » d'Arduino ou en important la librairie en format zip.

Installation depuis le store :

1) Allez dans l'onglet Croquis ou Sketch
2) Naviguez jusqu'au sous-onglet Inclure librairie ou Add librairie.
3) Cliquez sur le Gestionnaire de bibliothèque (library manager)

Depuis ce menu, il est possible d'installer, mettre à jour et supprimer les bibliothèques.

Installation depuis le un fichier zip :

1) Télécharger la librairie en format zip depuis github
2) Suivre l'étape 2 et 3 ci-dessus
3) Cliquez sur Ajoutez la librairie ZIP (Add library from ZIP)

Annexe 5 : Installation de l'Arduino UNO et du NodeMCU

Vous trouverez en annexe 1, les liens de téléchargements des programmes pour l'Arduino UNO et le NodeMCU. Sur chacune des pages Github, il y a la liste des dépendances requises (avec les liens de téléchargement) à installer avec l'Arduino IDE. Certaines dépendances ne sont pas sur le « store » et devront donc être installées en mode ZIP.

Installation pour l'Arduino UNO :

1) Connectez l'Arduino UNO au PC.
2) Allez dans l'onglet Outils et naviguez jusqu'au sous-onglet Type de carte.
3) Sélectionnez la carte Arduino/Genuino Uno
4) Naviguez jusqu'au sous-onglet Port et sélectionnez celui qui est relié à l'appareil.
5) Cliquez sur la flèche pointant sur la droite (Téléveser)

Installation pour le NodeMCU :

1) Connectez le NodeMCU au PC.
2) Allez dans l'onglet Outils et naviguez jusqu'au sous-onglet Type de carte.
3) Cliquer sur Gestionnaire de carte
4) Entrez le texte ESP8266 dans le champ de recherche et installez la dernière version de la carte ESP8266
5) Retourner dans l'onglet Type de carte et sélectionnez NodeMCU 0.9 (ESP-12 Module)
6) Naviguez jusqu'au sous-onglet Port et sélectionnez celui qui est relié à l'appareil.
7) Cliquez sur la flèche pointant sur la droite (Téléverser)

Sécurisation d'un réseau d'IoT avec le protocole Oauth2.0

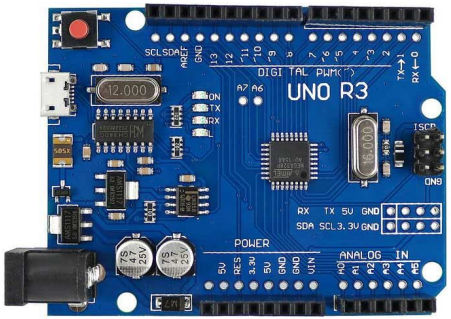
Sylvain Muller, Travail de Bachelor 2018

h e g
Haute école de gestion
Genève

Sommaire

- Introduction aux IoT
- Les IoT utilisés
- Introduction au framework Oauth 2.0
- Obtention de l’autorisation
- Introspection de jetons d’accès
- Preuve de la possession
- Mes contributions
- Démonstration
- Conclusion
- Questions

Introduction aux IoT



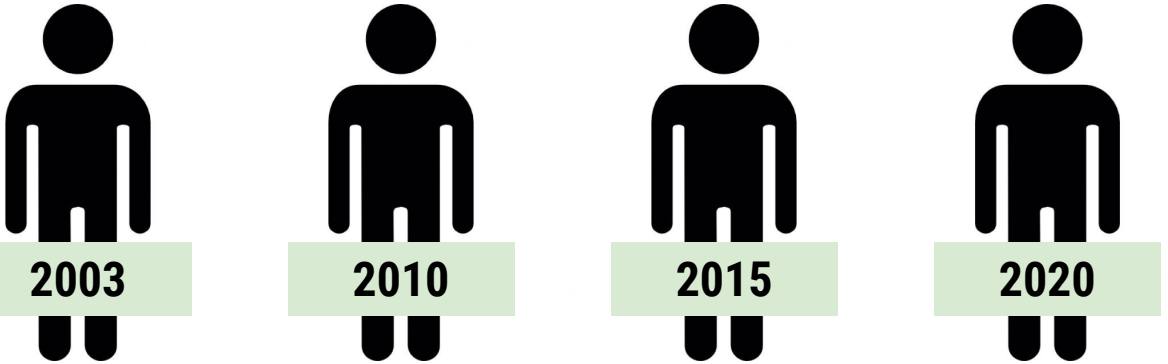
Définition

Wikipédia : **l'extension d'Internet** à des **choses** du monde physique.


>APPS
14:03

Forte croissance : nombres

	2003	2010	2015	2020
Population mondiale	6.3 bi	6.8 bi	7.2 bi	7.6 bi
IoT	500 mi	12.5 bi	25 bi	50 bi

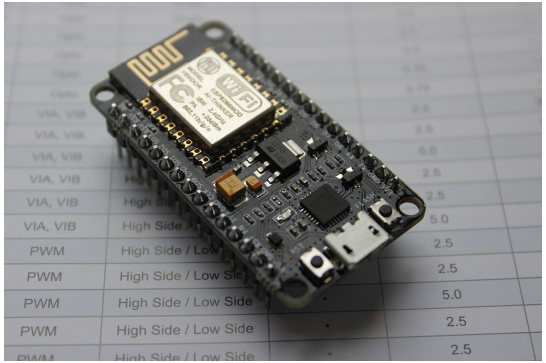


Principales vulnérabilités

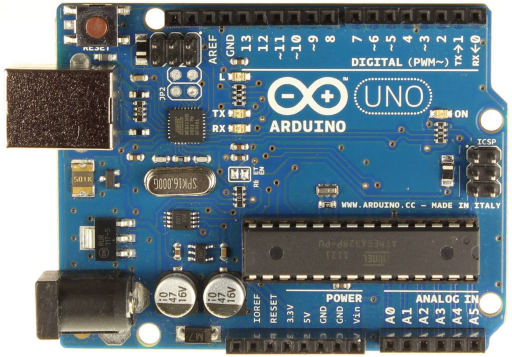
- Communication non sécurisée
- Mot de passe “hard codé” ou trop faible
- Driver ou firmware très ancien, pas de mise à jour auto

Les IoT utilisés

NodeMCU



Arduino UNO



Raspberry Pi 3B+

Les IoT sont des ressources de l’Internet

Sécuriser les **IoT** comme n’importe quel **service web**

Introduction au framework Oauth2.0

- Protocole industriel pour gérer les autorisations
- Standard développé par l'IETF (groupe de travail Oauth)
- Initié par Blaine Cook (Twitter) et Chris Messina (avocat américain dans le domaine des logiciels libres)

Les standards

Oauth 2.0 Core

RFC 6749 : Framework

RFC 6750 : Bearer Tokens

RFC 6819 : Security

Token Management

RFC 7662 : Token Introspection

RFC 7009 : Token Revocation

RFC 7519 : JSON Web Token

Other extensions

RFC 7521 : Assertions

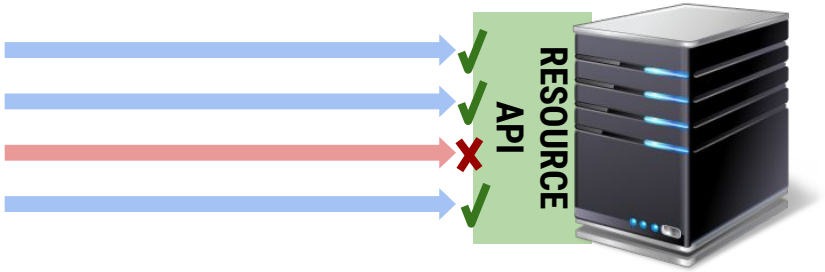
RFC 7522 : Bearer Assertion

RFC 7523 : JWT Bearer Assertion

https://oauth.net/2/

Objectifs principaux de Oauth 2.0

Protéger une **ressource**



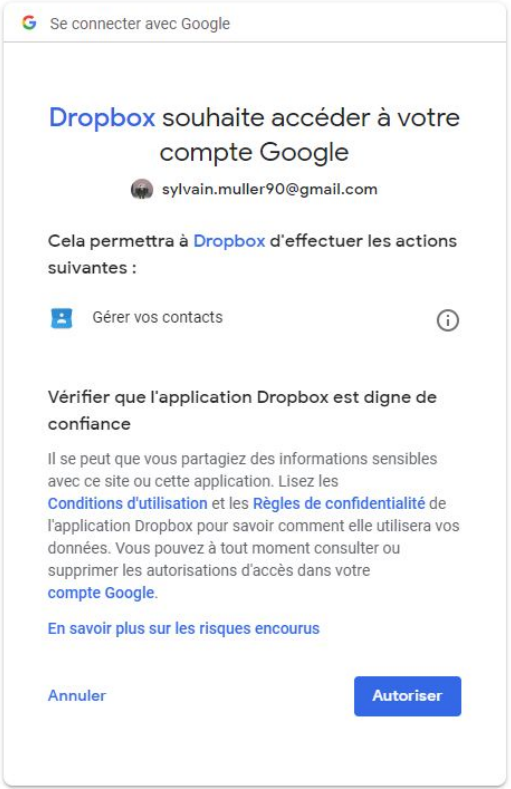
Accorder un **accès limité** à un **client** tiers

Un accès limité

- Limité dans le temps
- Autorisé par le propriétaire de la ressource
- Accès seulement au strict nécessaire

Concrètement

C’est **l’utilisateur** qui gère les **autorisations d’accès** à ses **données**



Garantie souhaitée

- Le client et la ressource ne sont jamais en possession des identifiants du “resource owner”
- L’autorisation délivrée est protégée (authentifié, intégrité et ou confidentialité)

Jetons d’accès

Bearer Token : chaîne de caractères opaque pour les clients

- JSON Web Token (pour le prototype)

JSON Web Token (JWT)

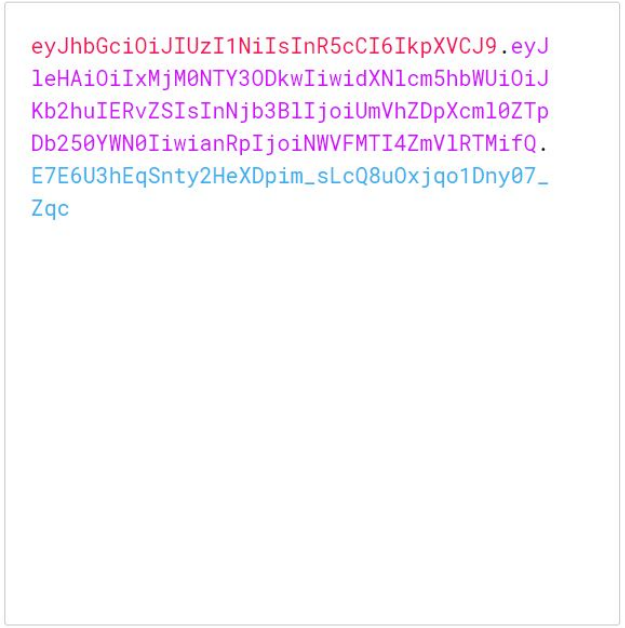
- Structure basée sur du JSON
- Contient des “claims”
- Protégé
  - Signé : MAC
  - Encrypté : Clé de chiffrement

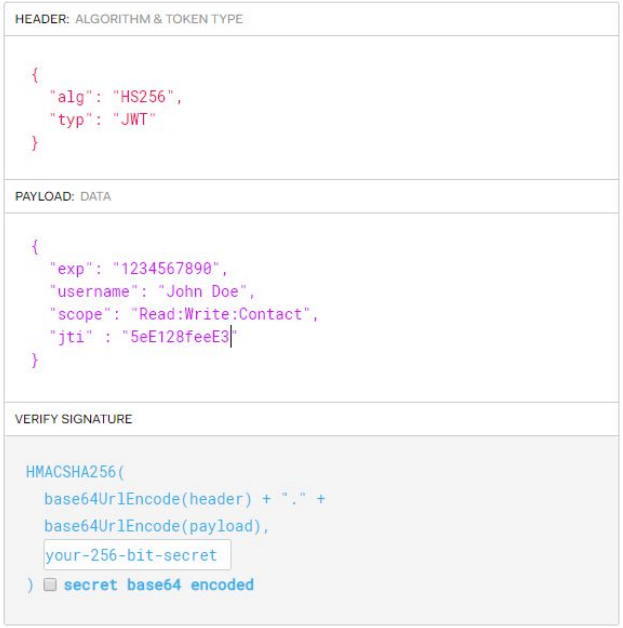
JSON Web Signature / JSON Web Encryption

Deux structures utilisées dans le prototype pour les JWT :

- JWS : HMAC avec SHA-256 (HS256)
- JWE : chiffré avec AES-CBC et signé avec SHA-2 (HS512)
    - Algorithme symétrique

JSON Web Token signé

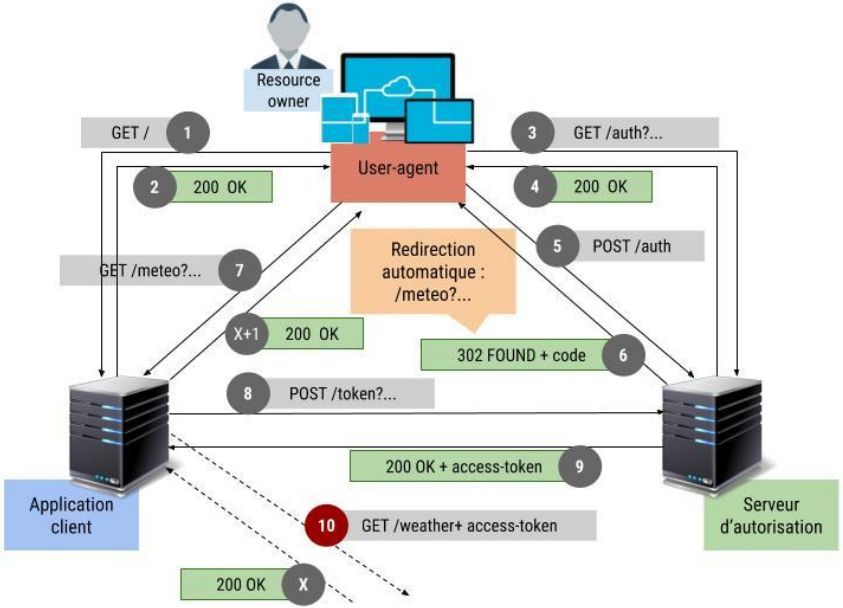




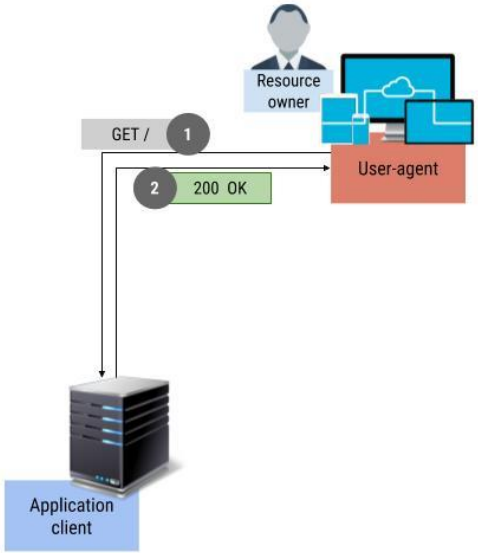
Obtenir une autorisation

- Authorization Code Grant
- Implicit Grant
- Client Credentials Grant
- Resource Owner Password Credentials Grant

Authorization Code Grant

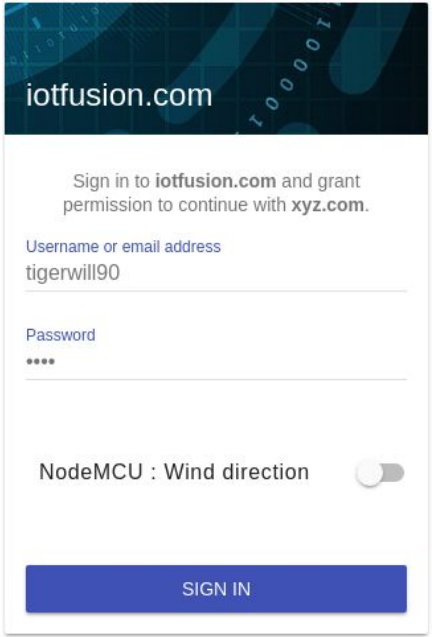


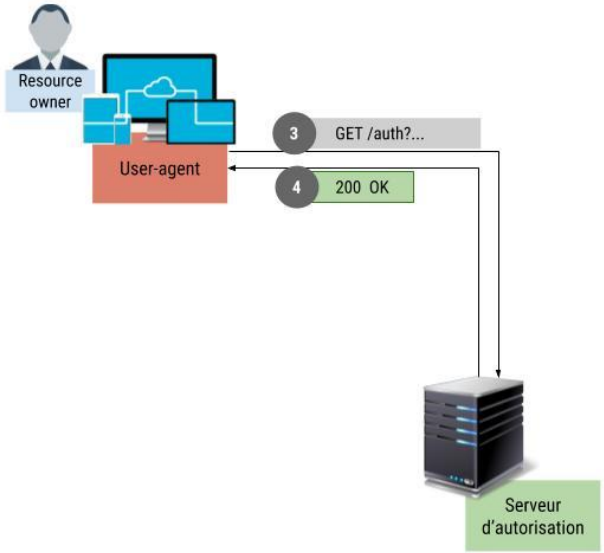
Authorization Code Grant



1. L'utilisateur arrive sur un site web tiers
2. L'application tiers lui demande de se connecter avec Oauth 2.0

Authorization Code Grant

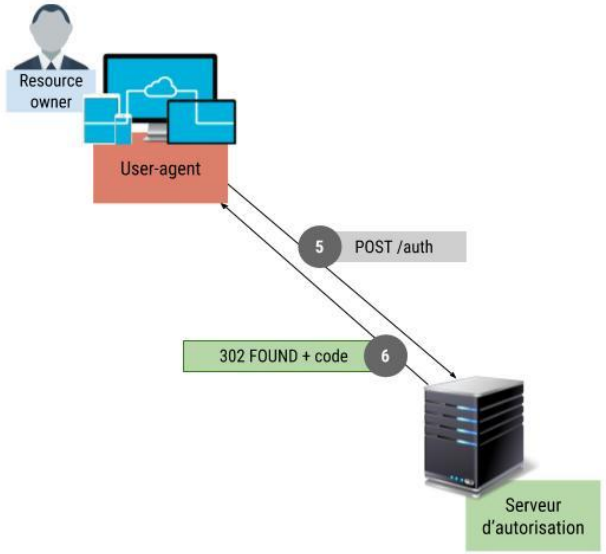




Authorization Code Grant

- L'utilisateur s'authentifie et accorde les autorisations
- Il est redirigé automatiquement par son navigateur

Implémentation :
Code : JSON Web Token (JWE)
Expiration : 10 minutes
Autre :
- Non réutilisable
- Ressource cible et scope
- Clé stocké en cache (Memcached)



Authorization Code Grant

Génération de la JWK

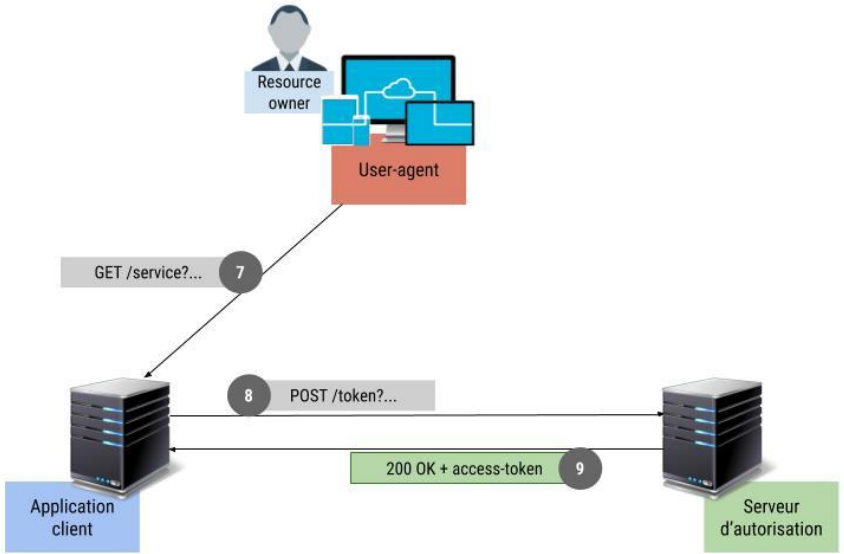
On stocke la clé en cache

Génération du payload

Génération de la JWT

```php
$jweKey = JWK::create([
    'alg' => 'A256KW',
    'enc' => 'A256CBC-HS512',
    'kty' => 'oct',
    'use' => 'enc',
    'k' => $this->generator->generateString(self::JWE_KEY_LENGTH, self::TOKEN_CHAR_GEN),
    'kid' => $kid,
    'key_ops' => ['encrypt', 'decrypt']
]);

$mc->set($kid, $jweKey, self::JWK_EXPIRATION);

$payload = [
    'aud' => $client->getClientName(),
    'exp' => time() + self::JWT_EXPIRATION,
    'iat' => time(),
    'iss' => getenv('APP_NAME'),
    'sub' => 'authorization_code',
    'jti' => $this->generator->generateString(self::NONCE_LENGTH, self::TOKEN_CHAR_GEN),
    'jwt' => [
        'aud' => $resource->getAudience(),
    ]
];

if (null !== $params['scope']) {
    $payload['jwt']['scope'] = implode(' ', $params['scope']);
}

try {
    $jwt = $this->joseHelper
        ->setJwk($jweKey)
        ->createToken($payload);
} catch (\Exception $e) {
    throw $e;
}
```

Authorization Code Grant



- Le client récupère le code d'autorisation (dans les paramètres de requête)
- Authentification du client
- Il récupère le **jeton d'accès**, le **jeton de rafraîchissement** et la **clé partagée**

Implémentation :
Jeton d'accès: JSON Web Token (JWS)
Jeton de rafraîchissement : JWE
Expiration : 60 secondes
Autre :
- Non réutilisable
- Ressource cible et scope
- Clé stocké en cache (Memcached)

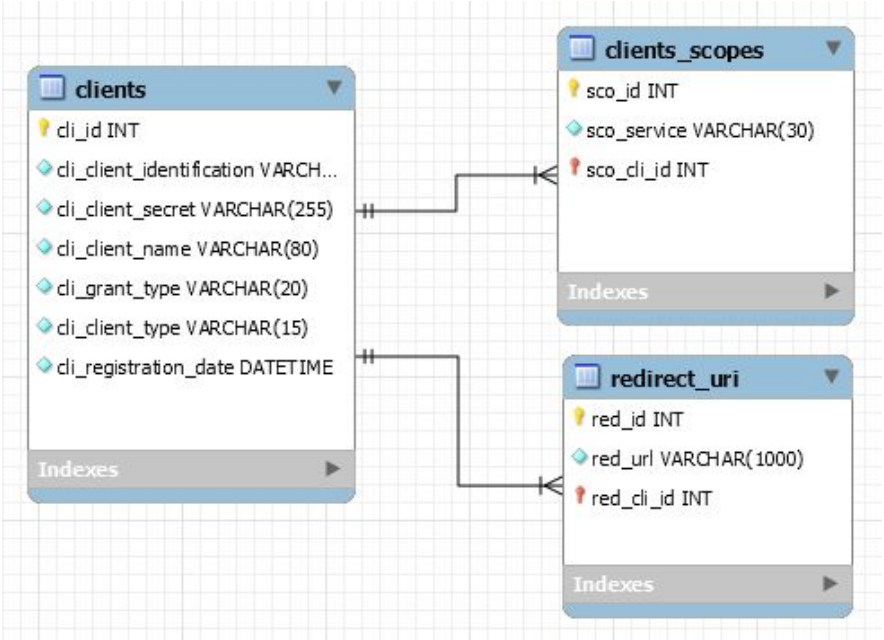
Authorization Code Grant

```php
// create shared key
$sharedKey = JWK::create([
    'alg' => $this->resource->getSharedKeyAlgorithm(),
    'kty' => 'oct',
    'kid' => $kid . '-s',
    'k' => $this->generator->generateString($this->resource->getKeySize(), self::TOKEN_CHAR_GEN),
    'key_ops' => ['encrypt', 'decrypt']
]);

$this->mc->set($kid . '-s', $sharedKey, self::JWK_EXPIRATION);
```

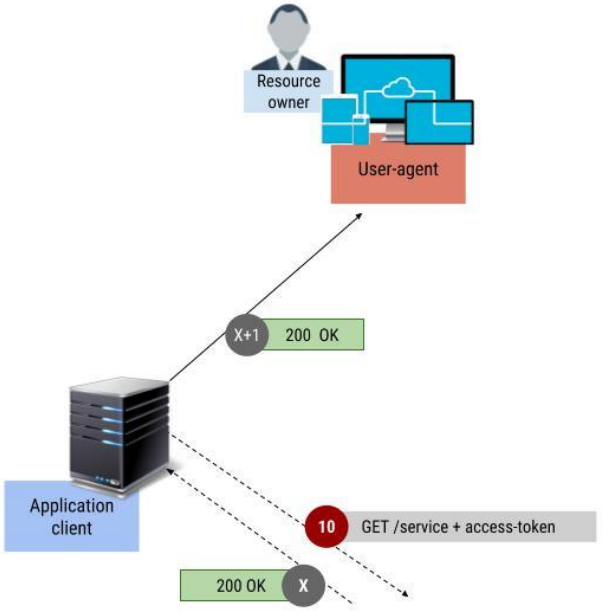
Algorithme de chiffrement supporté par la ressource

L'identifiant de la clé partagée est **identique** à celui de la clé du JWT

Authorization Code Grant



Authorization Code Grant



- Le client contacte la ressource avec le jeton d'accès
- Il retourne ensuite la valeur au propriétaire

Token Introspection

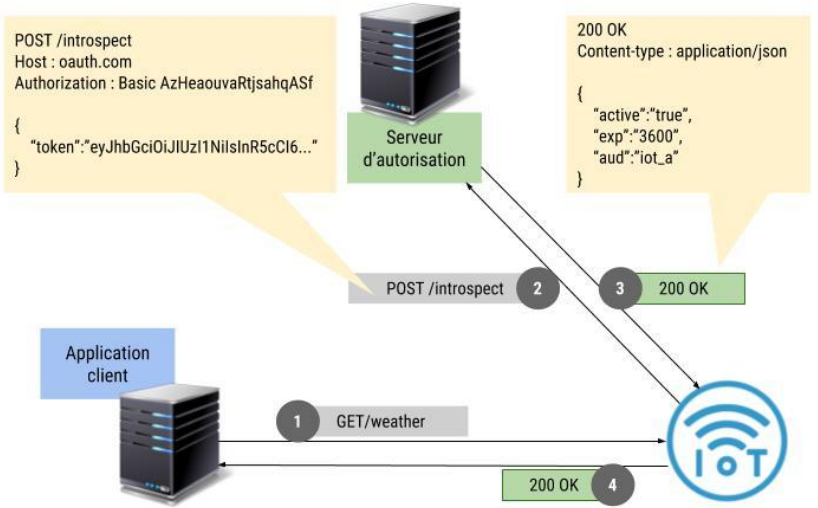
Pourquoi **déléguer** la responsabilité de la **validation** au **serveur d'autorisation** ?

Token Introspection

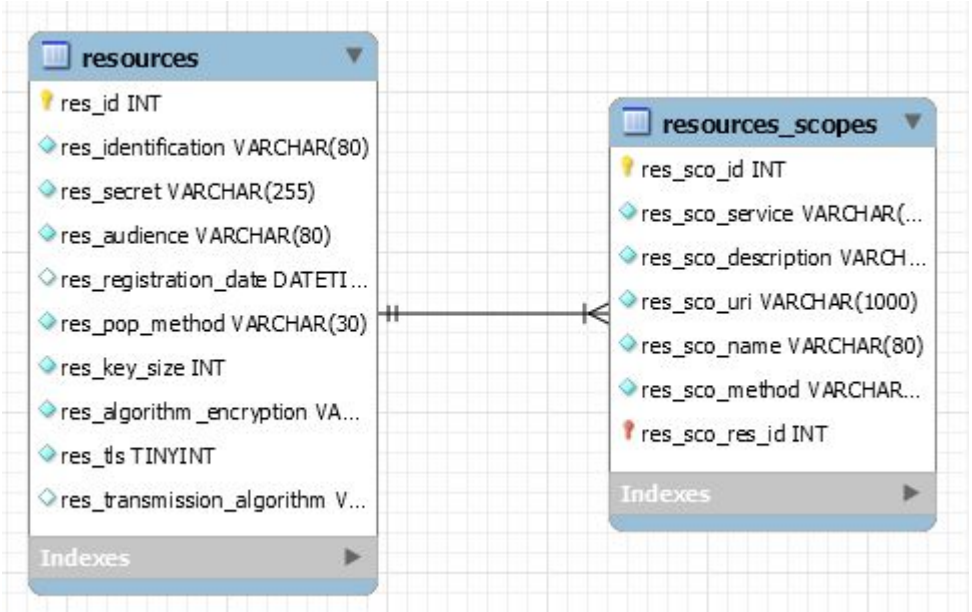
- La ressource reçoit une requête avec un jeton d'accès
- Elle le valide auprès de son serveur d'autorisation

Implémentation :

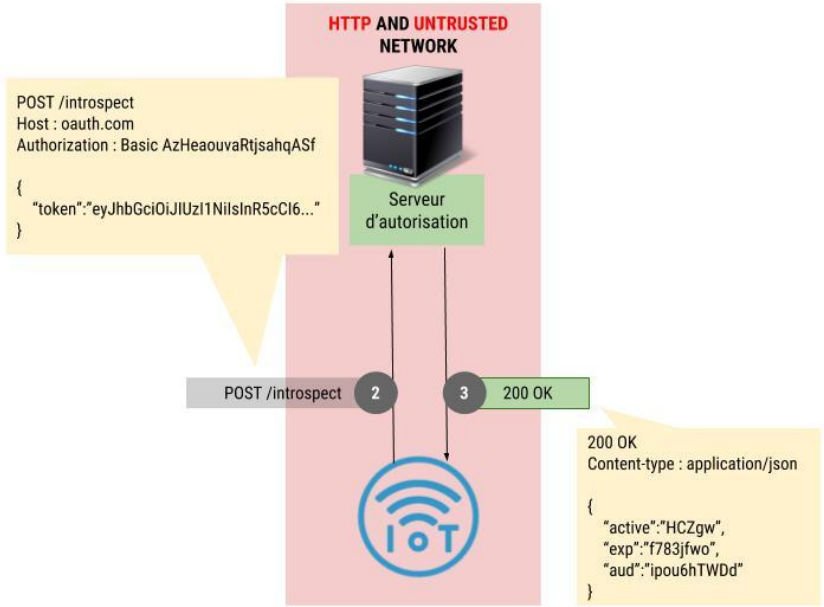
- Authentifie la ressource
- Le serveur d'autorisation retrouve la clé avec le **kid**
- Il valide tous les claims (sauf le scope)
- Stocke en cache le jti (nonce) jusqu'à échéance du JWT



Token Introspection



Token Introspection sans support de TLS

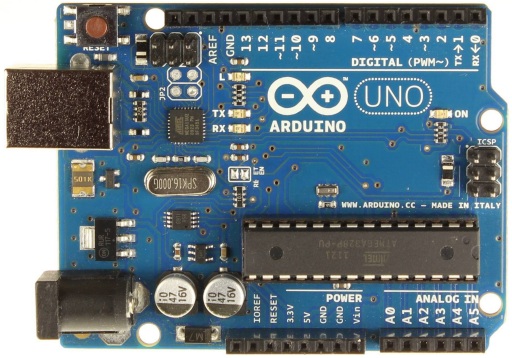


- Chiffre les valeurs retournées, idéalement avec un mode de d'opération chaîné (CBC) avec le "resource_secret"

Vulnérabilité du prototype :

- Aucune assurance de communiquer avec la bonne entitée

Token Introspection sans support de TLS



Solution :

- Le nonce (jti)

- Pas assez de **mémoire** pour stocker et déchiffrer le nonce

Proof of Possession

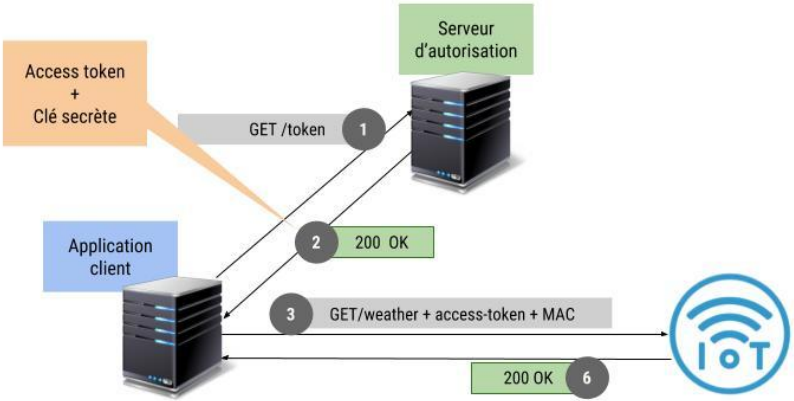
Un jeton d’accès **remplace** le traditionnel **identifiant** et **mot de passe**

Il **n’identifie** pas et peut donc être **volé**

Objectif du PoP

Prouver à une **ressource protégée** que le jeton d'accès **n'a pas été subtilisé**

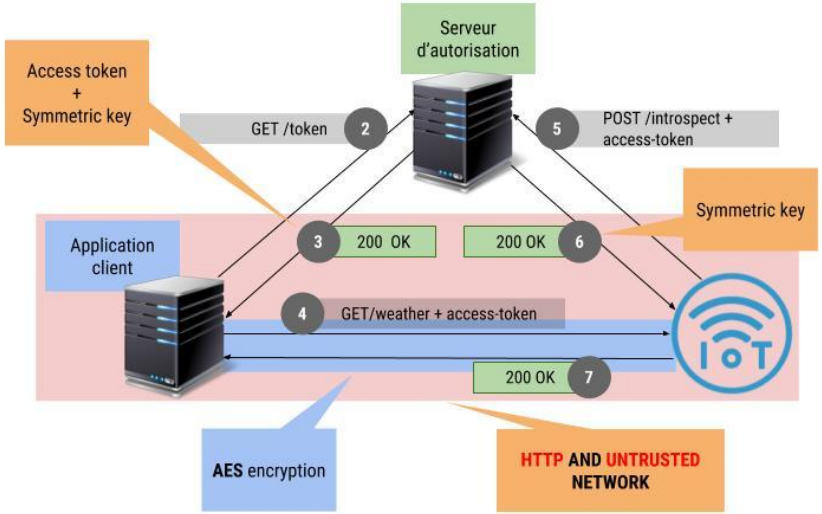
Flux du PoP (approche symétrique)



La ressource peut obtenir la clé secrète :

- Extraction depuis le jeton d'accès (la clé doit être chiffrée)
- Exécute une requête d'introspection pour récupérer la clé
- Partage la même base de données que le serveur d'autorisation

Proof of Possession sans TLS



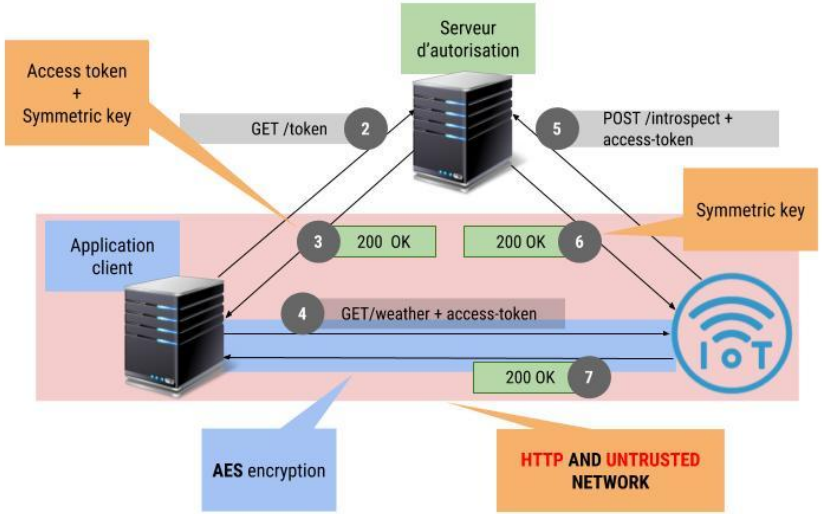
PoP avec le Token Introspection :

- Communication chiffrée et donc sécurisée
- La preuve de la possession est faite

Vulnérabilité :

- Man in the middle
- Replay attack

Proof of Possession sans TLS



Solution :

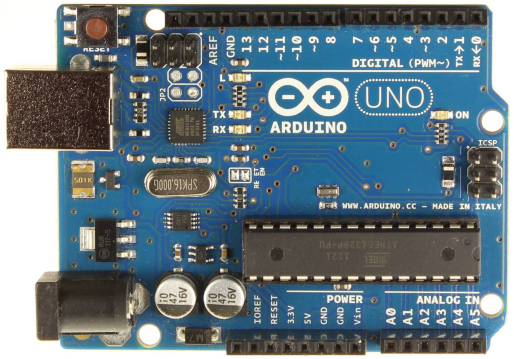
- Le nonce (jti)
- Expiration

Implémentation :

3 claims obligatoires pour chaque JWT

- Jti : nonce
- Exp : expiration
- Aud : audience

Problèmes rencontrés



Problèmes principaux :

- Pas de TLS
- Manque de mémoire
- Librairie REST incomplète
- Difficile de trouver une librairie AES fonctionnelle
- Mode d'opération ECB

Implémentation :

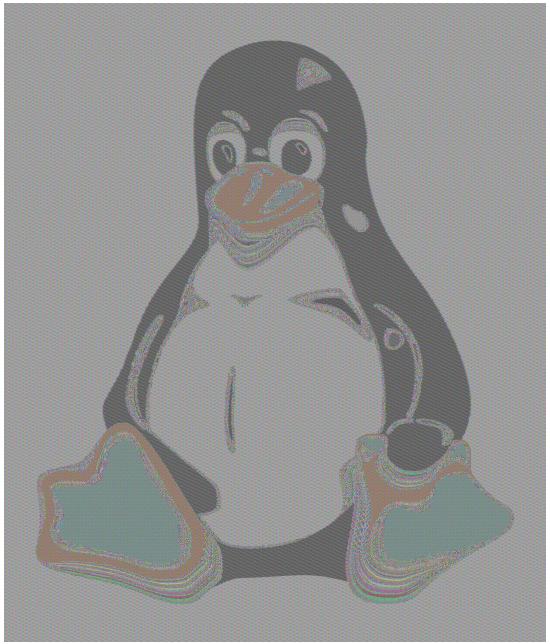
- Créer un Sketch pour l'Arduino UNO
- Fork de la librairie REST et modification
- Limiter la taille des buffers
- Limiter la taille des jetons d'accès
- Générer une nouvelle clé symétrique à chaque requête

Mode d’opération de cryptographie

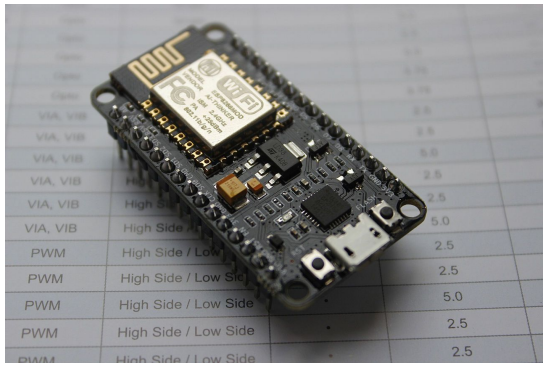
- ECB : Dictionnaire de code
  - Chaque bloc est chiffré séparément
- CBC : Enchaînement de blocs
  - On utilise le bloc précédent pour chiffrer le suivant

The ECB PENGUIN

Deux blocs **égaux avant** chiffrement le seront aussi **après**



Problèmes rencontrés



Problèmes principaux :

- Impossible de vérifier l'empreinte digital SHA-1 lors de requêtes https
- Impossible de faire des requêtes POST pour l'introspection
- Incompatibilité des librairies AES de l'Arduino

Implémentation :

- Créer un Sketch pour le NodeMCU
- Utilisation d'une librairie client REST tiers pour faire des requêtes https
- Introspection avec le token en en-tête

Mes contributions (Oauth 2.0 RFC 6749)

Un package php pour orchestrer l'obtention d'un jeton d'accès ou d'un code d'autorisation avec les types de réponses "Implicit" (section 4.2) et "Authorization code" (section 4.1)

- Requête d'autorisation (PSR-7) en entrée (section 4.1.1 et 4.2.1)
- Validation de la requête
- Authentification du propriétaire de la ressource
- Réponse/erreur standardisée (section 4.1.2 et 4.2.2)

Mes contributions (Oauth 2.0 RFC 6749)

Un package php pour orchestrer l'obtention d'un jeton d'accès à partir d'un jeton de rafraîchissement ou d'un code d'autorisation.

- Requête d'accès (PSR-7) en entrée (section 4.1.3 et 6)
- Validation de la requête
- Authentification du client
- Réponse/erreur d'accès standardisée (section 4.1.4)

Mes contributions (Oauth 2.0 RFC 7662)

Un package php paramétrable pour orchestrer l'introspection de jeton d'accès (Introspection Token)

- Requête d'introspection (PSR-7) et JWKset en entrée (section 2.1)
- Validation de la requête
- Validation des paramètres d'autorisations (claims)
- Support PoP réponse (non standardisée)
- Réponse/erreur d'introspection standardisée (section 2.2 et 2.3)

Mes contributions (Arduino UNO)

Modification d’une librairie REST orientée route compatible avec les “Bearer token”.

- Extraction du jeton d’accès dans l’en-tête HTTP
- Support de la réponse “404 not found”
- Spécialement optimisée pour la démonstration

Démonstration

Conclusion

- Travail très intéressant
- Un protocole d’avenir pour les IoT

Questions