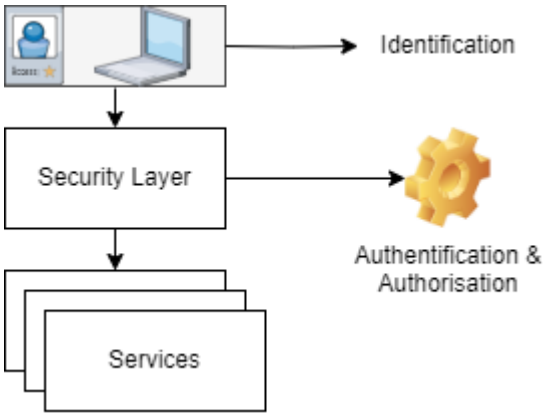
# Modes d'authentification

## 1. Introduction

Trois étapes sont nécessiares pour pouvoir utiliser des applications ou des sites sécurisés:

- L'identification : l'utilisateur se présente et affirme qu'il est le bon utilisateur
- L'authentification : l'utilisateur prouve son identité via mot de passe, empreinte, code SMS, ...
- L'authorisation : l'utilisateur, une fois authentifié, aura l'accès à quelques ressources selon ces droits




## 2. Authentification via mot de passe

Dans ce mode basique d'authentification, l'utilisateur saisie ces uniques login (username) et mot de passe (password) pour acquérir les accès à des ressources protégées dans une application ou un site web.

Ces informations d'identification saisies seront checker à travers le système par rapport aus informations utilisateur stockées, et s'ils correspondent bien à ces derniers, l'utilisateur pourra accéder au ressources demandées.

le faite que cette méthode d'authentification est la plus utilsée dans le monde de logiciel, ne l'enpêche pas d'avoir beaucoup de limites !

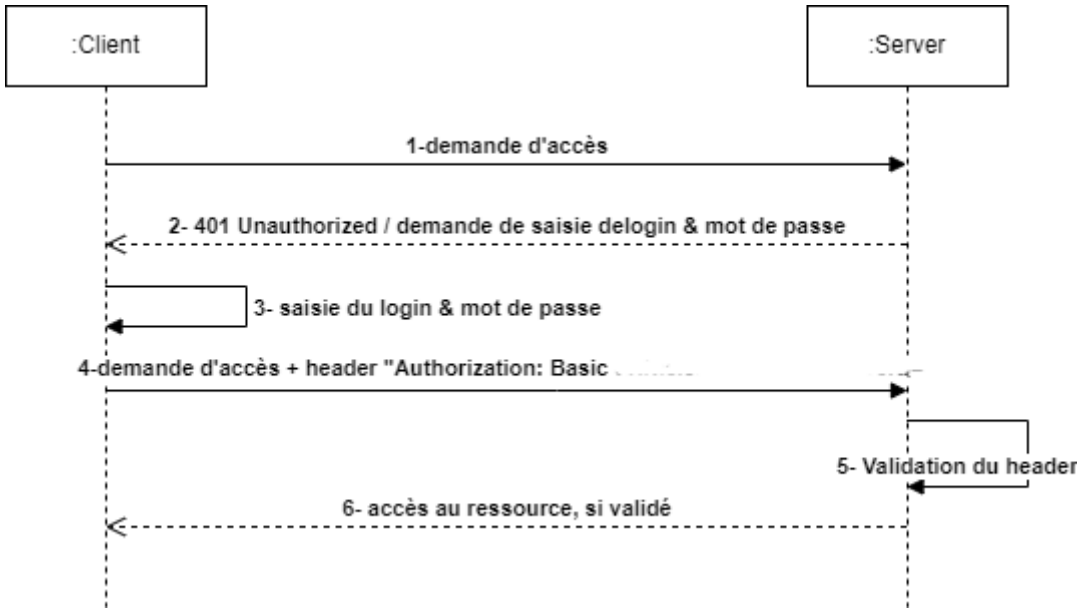
- risque d'oublie des mot de passe par les utilisateurs
- gestion très dure de plusieurs couples login/mot de passe pour chaque application/site utilisé
- nécessité de mesure de sécurité très poussées pour les systèmes basés sur ce mode pour éviter les vulnérabilités aux attaques

Pour contourner tous ces limites, les nouveaux systèmes implémentent tel que l'authentification multi-facteur, l'authentification via token, ...etc.

Dans ce qui suit, on détaillera les mécanismes d'authentification via mot de passe avant d'attaquer les autres alternative dans les chapitres qui suivent.

### a. HTTP Basic Authentication

Les informations d'identifications (login et mot de passe) dans ce mécanisme sont encodées via l'algorithme Base64 et ajoutées au header http "**Authorization: Basic**" le processus détaillé de ce mécanisme est le suivant :




- 1. Demande d'accès à une ressource dans le serveur
- 2. Si l'utilisateur n'est pas encore authentifié, la réponse sera une erreur HTTP '**401 Unauthorized**' avec un header "**WWW-Authenticate: Basic**" pour demander une "**HTTP Basic Authentication**"
- 3. l'utilisateur saisit alors son login et mot de passe. Ces données seront combinés dans une seule chaine de caractère sous le format "**username:password**", encodée et ajoutée au header "**Authorization**" : exp : exp: "**Authorization: Basic dXNlcl5hbWU6cGFzc3dvclQ=**"
- 4. le client renvoie de nouveau la requête dedans le header des informations d'identification.
- 5. le serveur décode le header d'authorisation et check si ça correspond aux données utilisateur dans la base de données ou dans un serveurs d'authentification tièrce.
- 6. si le header est validé, le serveur donne accès au ressource demandée. Sinon, l'erreur HTTP '**401 Unauthorized**' est levée.

Notez que ce mécanisme d'authentification n'est presque plus utilisé dans les applications/sites modernes car:

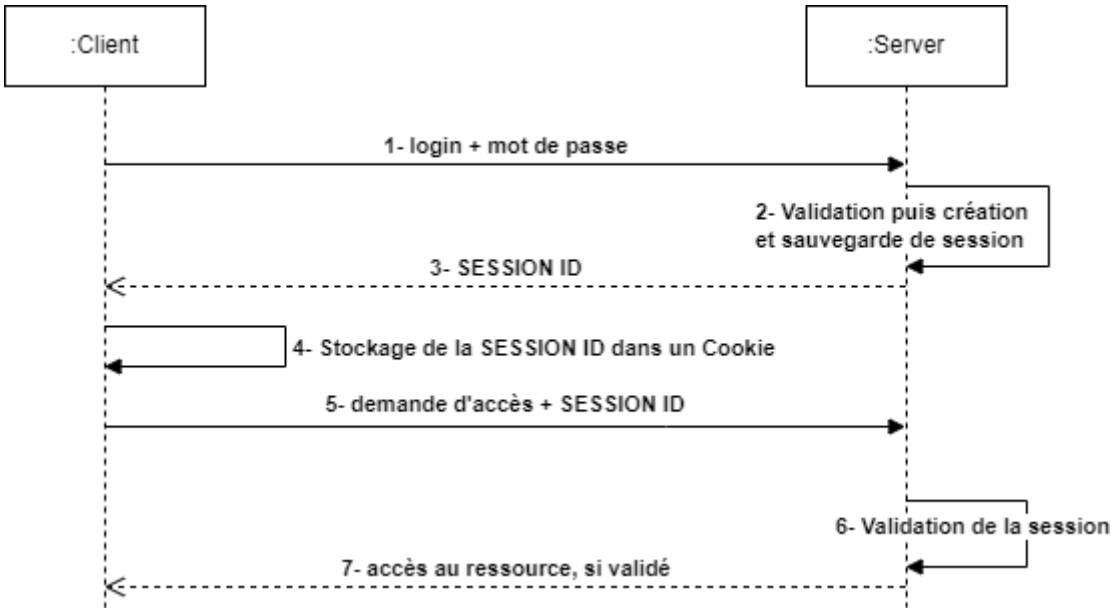
- les logins & mots de passe transitent dans le réseau encodé en Base64, et leurs interceptions et décodages par des malveillants sont très simple
- les logins & mots de passe sont re-envoyés pour chaque requête de ressource protégée!

## b. Session-Cookie Authentication

Ce mécanisme d'authentification incite à créer (au niveau serveur), pour un utilisateur authentifié, un identifiant unique (le "**session ID**") partagé entre le serveur et le client (via "**Cookie**") et qui a une durée de vie limité (session timeout).

Ceci évitera de re-envoyer à chaque requête les informations sensibles login et mot de passe, mais ceux-ci devront être envoyées quand-même initialement pour pouvoir créeer la "**Session ID**".

le processus détaillé de ce mécanisme est le suivant :



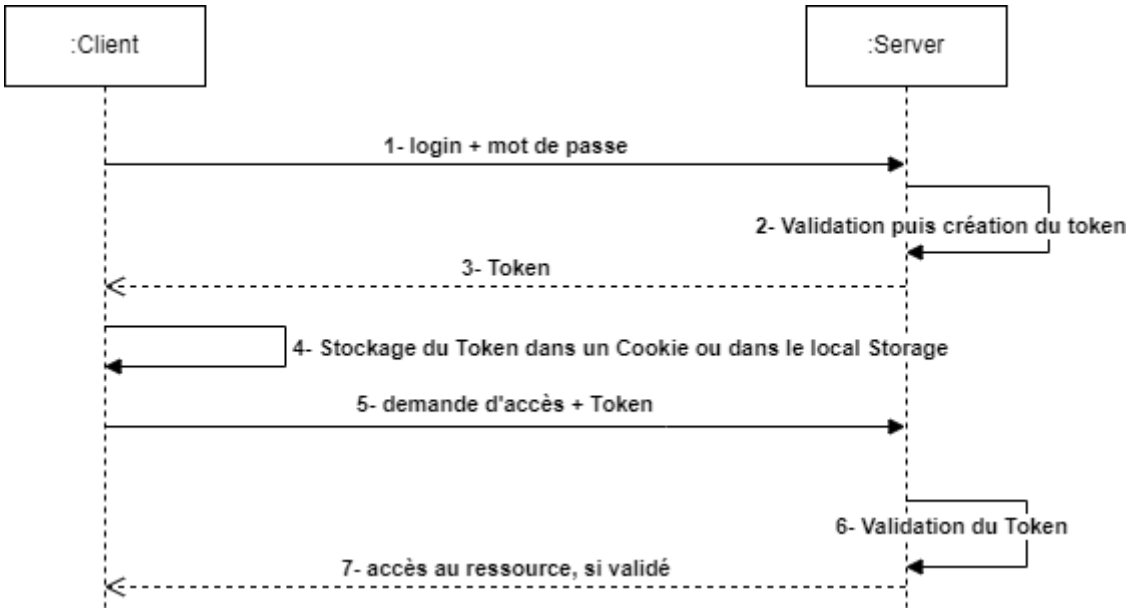

- 1. l'utilisateur soumit son login et mot de passe au serveur
- 2. le serveur check si ça correspond aux données utilisateur dans la base de données ou dans un serveurs d'authentification tièrce. Si ça correspond, le serveur créé l'unique "**Session ID**" et stocke la session associée qq part dans le serveur (mémoire, BD, serveur de session)
- 3. le serveur renvoie le "**Session ID**" au client typiquement via un header "**Set-Cookie**"
- 4. le client stocke le Cookie de session.
- 5. le client accède aux ressources protégés en envoyant le cookie de session dans les header de ses requêtes.
- 6. le serveur check le coookie envoyé dans les headers et le compare au Session ID sauvegardé de son coté avant de servir le client.
- 7. si OK, le serveur donne accès au ressource demandée.

### c. Token-Based Authentication

Ce mécanisme d'authentification incite à créer (au niveau serveur), pour un utilisateur authentifié, un "**token**" (Jeton) d'une durée déterminée à renvoyer au client.

tout comme le Session-Cookie, ceci évitera de re-envoyer à chaque requête les informations sensibles login et mot de passe, mais ceux-ci devront être envoyées quand-même initialement pour pouvoir créér le "**Token**".

le processus détaillé de ce mécanisme est le suivant :




- 1. l'utilisateur soumit son login et mot de passe au serveur
- 2. le serveur check si ça correspond aux données utilisateur dans la base de données ou dans un serveurs d'authentification tièrce. Si ça correspond, le serveur créé le "**Token**"
- 3. le serveur renvoie le "**Token**" au client
- 4. le client stocke le "**Token**" dans un Cookie ou dans Locale Storage.
- 5. le client accède aux ressources protégés en envoyant le "**Token**" avec ses requêtes.
- 6. le serveur check le "**Token**" envoyé avant de servir le client.
- 7. si OK, le serveur donne accès au ressource demandée.

Notez les avantages suivante de ce mécanisme par rapport au mécanisme précédent :

- le "**Token**" n'est pas stocké au niveau serveur, il est stocké coté client seulement. Et donc y'a un gain d'espace de stockage et de scalabilité coté serveur.
- la durée de vie d'un token peut être beaucoup plus longues que la durée de vie d'une session.
- ce mécanisme est une bonne approche pour résoudre les problèmes Cross-domain et implémenter le SSO.

Par contre, deux limites pricipales sont identifiées pour ce méanisme:

- le risque de récupération des "**Token**" par des utilisateurs malveillants !
- les tokens sont plus suceptibles aux attaques XSS et CSRF puis qu'ils ne sont stockés que coté client! ==> Il faut donc s'assurer que toutes les les précautions et mesures de sécurité sont bien mise en oeuvre !

### d. JSON-Web Token (JWT)

JWT est un standard qui étend le "**Token-Based Authentication**". Il fournit un moyen compact pour transférer des données entre les parties intervenantes sous forme d'objet JSON signé par une signature éléctronique.

Le JWT est composé de trois parties:

- le Header : contient des informations qui concernent le token lui même tel que par exemple son type (ex: jwt) et l'algorithme de signature (ex: HS512)

- le payload : contient des informations sur l'utilisateurs tel que son nom, son login, ces roles, ...etc.
- la signature : générée en utilisation une clè secrête au niveau serveur

Exemple de génération d'un JWT : **TODO** ref: https://fr.wikipedia.org/wiki

# 3. Authentification sans mot de passe

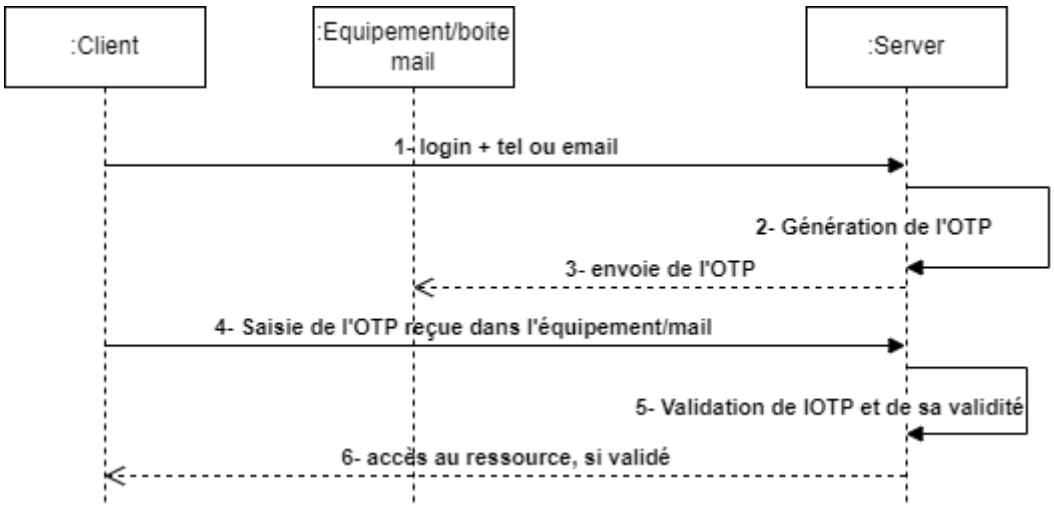
Jusqu'au là, on a vue des méthodes d'autentification qui oblige l'utilisateur à créer et se rappeler d'un mot de passe ! Cependant, il y'a d'autres méthodes qui peuvent identifier un utilisateur sans mot depasse ! ces méthodes se basent sur l'identification

- d'un équipement en possession de l'utilisateur (téléphone par exemple)
- d'une information unique à l'utilisateur (son empreinte biométrique par exemple)

## a. One-Time Password (OTP)

l'OTP est un code envoyé à l'utilisateur via un équipement (SMS au téléphone par exemple) ou un progiciel (une boite mail par exemple) pour s'identifier. Ce code est généralement utilisable une seule fois et dans une fenètre de temps limité pour qu'il ne soit pas piraté et réutilisé.

le processus détaillé de ce mécanisme est le suivant :




1. l'utilisateur veut se loguer a un site web et est invité à saisir son login, son numéro de Tél ou son email
2. le serveur génère un OTP avec un temps d'expiration
3. le serveur renvoie l'OTP à l'utilisateur et demande de le saisir dan le site web
4. le client saisie l'OTP dans le champs associé dans le site web et valide.
5. le serveur compare l'OTP saisie à l'OTP généré..
6. si OK, le serveur donne accès au ressource demandée

# 4. SSO (Single Sign On)

L'authentification unique ou Single Sign-On (SSO) est la fonction qui permet aux utilisateurs de s'authentifier une seule fois indépendamment du nombre de services et/ou applications sollicités. Ils peuvent alors accéder à leurs données en toute transparence, sans contrainte de ressaisie de nouveaux leurs paramètres d'accès à chaque invocation d'un nouveau service et/ou application. C'est aussi l'acronyme de Single Sign-Out (la

déconnexion unique) qui garantit le nettoyage des cookies de sessions pour tous les services/applications une fois déconnecté de l'un d'eux !

### a. OAUth2

Est un protocole de délégation d'accès ou autorisation. Le client s'authentifie auprès du serveur d'autorisation et obtient un jeton opaque appelé « jeton d'accès ». Ce dernier contient uniquement une référence aux informations utilisateurs et qui ne peut être décodé que par le serveur d'autorisation lui-même. Ce type de jetons est appelé « jeton de référence » et il est sûr de l'utiliser même dans le réseau public / Internet.

### b. OpenID Connect (OIDC)

C'est une extension de OAuth2 qui émet en plus du jeton d'accès un jeton d'identité qui contient des informations utilisateurs appelées « Claims ». Ceci est souvent implémenté par un JWT (JSON Web Token) signé par un serveur d'autorisation. Cela permettra de garantir la confiance entre le « serveur d'autorisation » et le client. Ce type de jetons est appelé « jeton par valeur » et il vaut mieux éviter de l'utiliser dans le réseau public / Internet.

Les jetons OAuth2/OpenID Connect mise en jeux sont généralement :

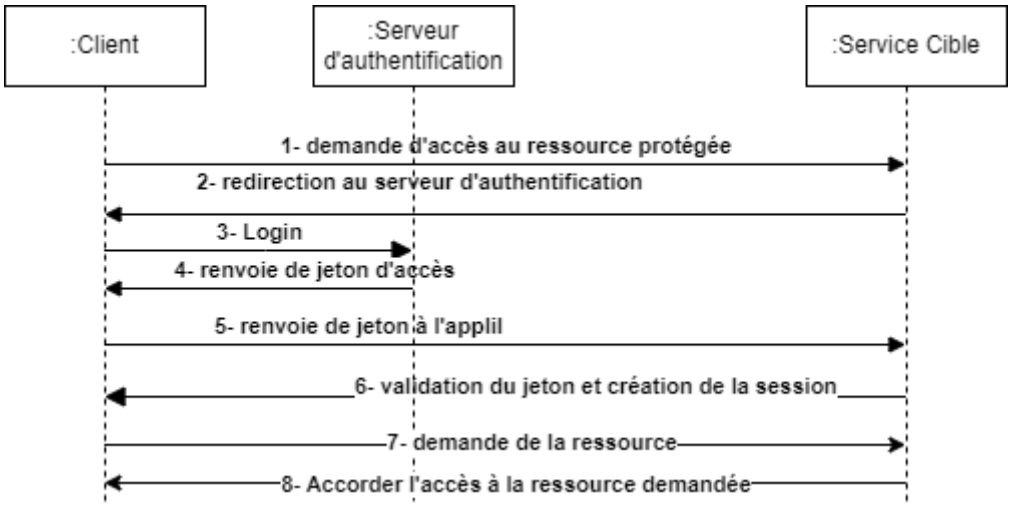
- "**id_token**" (jeton d'identité): Un jeton JWT émis par le serveur d'autorisation et consommé par le client. Les « claims » dans ce jeton contiendront des informations sur l'utilisateur afin que le client puisse les utiliser. Exemple de UCs (Authentification unique (SSO)): Un utilisateur se connecte à un service en utilisant ses identifiants de connexion pour un autre service. Le service d'authentification émet un jeton d'identité (id_token) qui contient des informations sur l'utilisateur, permettant au service client de l'authentifier sans avoir à saisir à nouveau ses identifiants.
- "**access_token**" (jeton d'accès): Un jeton JWT émis par le serveur d'autorisation et destiné à être consommé par la ressource. Exemple de UCs (Accès à des ressources protégées): Un utilisateur accède à une ressource protégée en utilisant un jeton d'accès (access_token) émis par le serveur d'autorisation. La ressource vérifie la validité du jeton d'accès avant d'autoriser l'accès à l'utilisateur.
- "**refresh_token**" (jeton de rafraîchissement): Il s'agit d'un jeton émis par « serveur d'autorisation » que le client doit utiliser lorsqu'il a besoin d'actualiser le id_token et access_token. Le jeton est opaque pour le client et ne peut être utilisé que par « serveur d'autorisation ». Exemple : Un utilisateur utilise un jeton de rafraîchissement (refresh_token) pour obtenir de nouveaux jetons d'identité et d'accès lorsque les jetons précédents ont expiré. Cela permet à l'utilisateur de rester connecté sans avoir à saisir à nouveau ses identifiants.

Ces jetons seront détruits à la suite de l'expiration de session ou la déconnexion volontaire par l'utilisateur. Ces jetons sont généralement signés pour garantir leurs intégrités. Pour valider le jeton par un service ou une application, plusieurs solutions sont envisageables selon le « serveur d'autorisation » choisie :

- Utilisation d'une clé publique : Cette méthode nécessite une configuration côté application/service pour utiliser la clé publique fournie par le serveur d'autorisation. L'avantage de cette méthode est qu'elle est sécurisée et ne nécessite pas d'appels HTTP supplémentaires pour valider les jetons. Cependant, elle nécessite une configuration initiale pour mettre en place la clé publique.
- Appel d'un service dédié dans le « serveur d'autorisation » : Cette méthode nécessite des appels HTTP supplémentaires à chaque validation de jeton, ce qui peut ralentir le processus. Cependant, elle ne nécessite pas de configuration initiale côté application/service.

- Utilisation d’une clé secrète : Cette méthode est considérée comme moins sûre que les autres méthodes, car la clé secrète doit être partagée entre le serveur d'autorisation et l'application/service. Si la clé secrète est compromise, la sécurité des jetons peut être compromise !

le processus détaillé de ce mécanisme est le suivant :




## C. Quelques bonnes pratiques d'utilisation de jetons OAuth2/OpenID Connect

- **Utiliser des jetons à durée de vie courte:** Les jetons d'accès et d'identité doivent avoir une durée de vie courte pour réduire les risques en cas de compromission. Les jetons de rafraîchissement peuvent avoir une durée de vie plus longue, mais doivent être stockés en toute sécurité.
- **Valider les jetons:** Les jetons doivent être validés avant d'être utilisés pour accéder à des ressources protégées. Les méthodes de validation incluent l'utilisation d'une clé publique, l'appel d'un service dédié dans le serveur d'autorisation ou l'utilisation d'une clé secrète. La méthode la plus sûre est l'utilisation d'une clé publique.
- **Utiliser des connexions sécurisées:** Les jetons doivent être transmis uniquement via des connexions sécurisées (HTTPS) pour éviter les interceptions.
- **Gérer les autorisations:** Les autorisations accordées aux jetons doivent être gérées avec soin pour éviter les accès non autorisés aux ressources protégées.