# UE 3&4 M3-3 Sécurité web

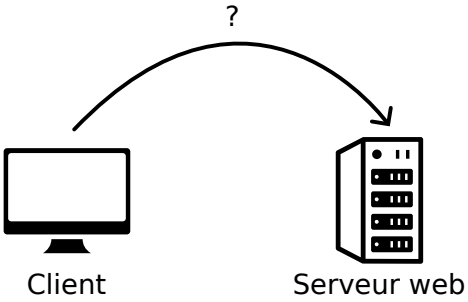
## Authentification des utilisateurs

Louis DEGUILLAUME
louis.deguillaume@u-bordeaux.fr

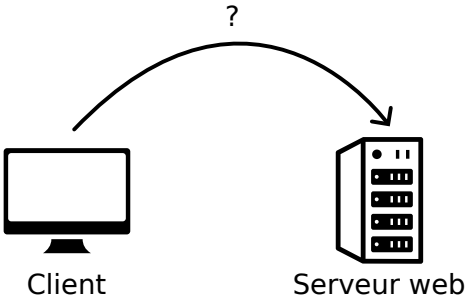


18 décembre 2022

# Comment protéger l'accès à une ressource ?

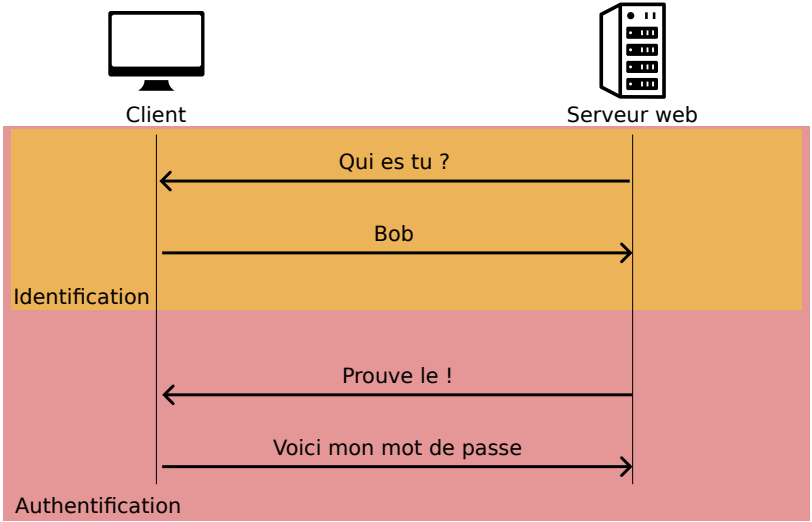


# Comment protéger l'accès à une ressource ?

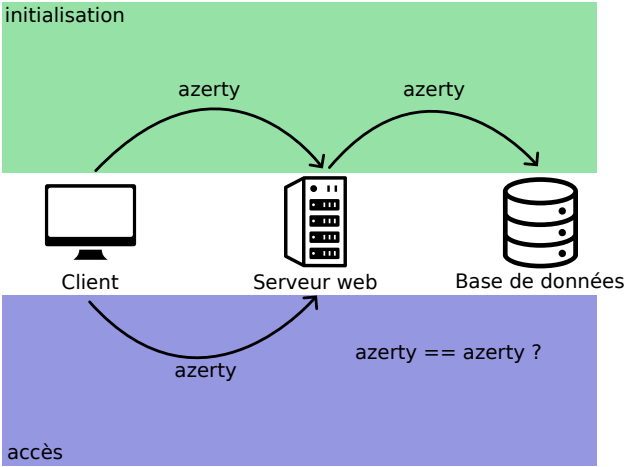



→ Une solution mise en place peut-être l'usage d'un mot de passe.

# Identification vs Authentification

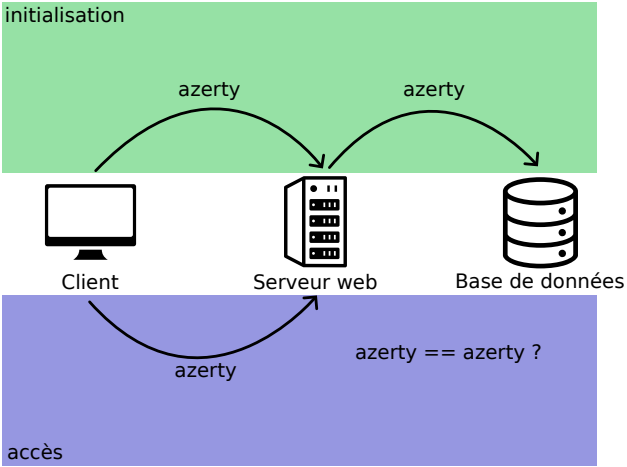


# Stocker le mot de passe dans une base de données ?




+ Authentification de l'utilisateur

# Stocker le mot de passe dans une base de données ?




\+ Authentification de l'utilisateur

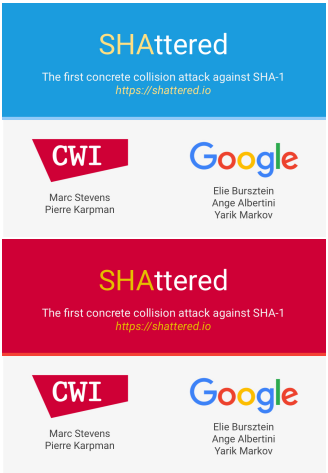
- Problème de sécurité en cas de fuite de base de données
- Confidentialité en cas de réutilisation de mot de passes

# Calcul d'empreinte

- À partir d'une chaîne de taille quelconque, on obtient un hash, ou condensat,
  - de taille fixe
  - unique : il n'est pas possible d'obtenir le même condensat à partir de 2 chaînes différentes
  - non réversible
- Principe mathématique :
  Soit $A$ et $B$ deux fichiers (ou chaînes)
  Soit $f$ une fonction de calcul d'empreinte
  $f(A) = A'$
  $f(B) = B'$
  Si $A' = B'$ alors $A = B$
  Si $A' \neq B'$ alors $A \neq B$

# Résistance des fonctions de calcul d'empreintes

- Un algorithme est considéré non sûr si :
  - cas d'une collision :
    $A' = B'$ et $A \neq B$
  - algorithme insuffisamment sûr :
    - il est possible de déduire A de A'
- Certains algorithmes ne sont plus considérés comme sûrs comme le MD5 et le SHA1 : il est possible de générer des collisions.
  - Ex : Deux fichiers pdf différents avec le même sha1
- Préférez SHA256 ou SHA512 par exemple.



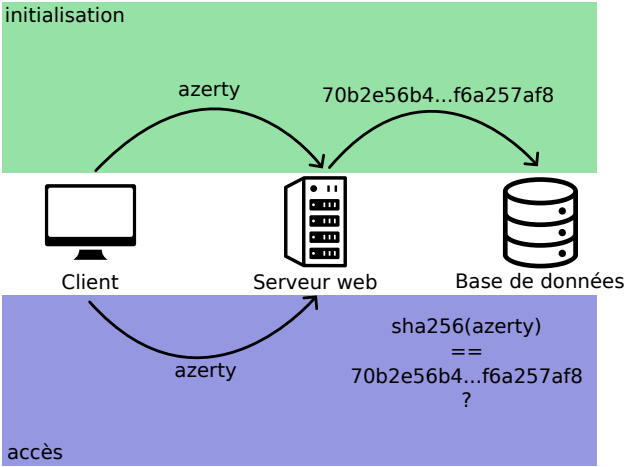

*Source : shattered.io*

# Calcul d'empreinte

```
$ echo 'AZERTY' | sha256sum
194b0cdc29ac7d6f42bc1886096f0319e977e70f88365348da649ed2255e1a30
```

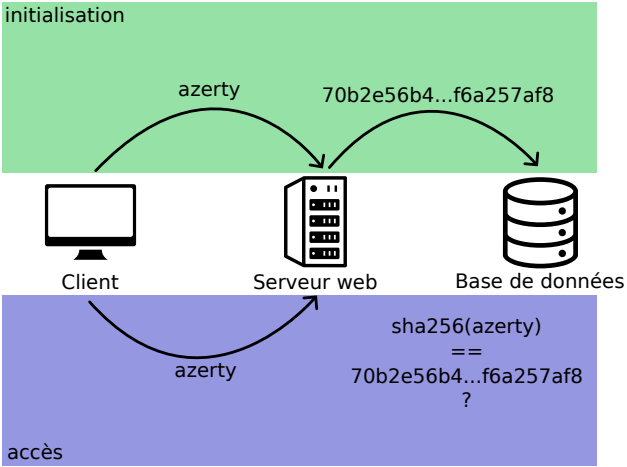
```
$ echo 'BZERTY' | sha256sum
78b87dad4b5a48f1e43d318599287ad57805c49fa039208d9de7e11f866c780a
```

# Stocker le mot de passe hashé dans une base de données ?




- Authentification de l'utilisateur
- Plus compliqué de retrouver le mot de passe... Mais pas impossible.

# Stocker le mot de passe hashé dans une base de données ?




+ Authentification de l'utilisateur
+ Plus compliqué de retrouver le mot de passe... Mais pas impossible.

- Problème de sécurité en cas de fuite de base de données
- Confidentialité en cas de réutilisation de mot de passes
  - Mot de passes courts
  - Rainbow tables

# Le salage

**Problème** : Deux mots de passe identiques ont la même empreinte

- si les mots de passe sont trop simples, il est facile de les retrouver à partir de l'empreinte
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

# Le salage

**Problème** : Deux mots de passe identiques ont la même empreinte

- si les mots de passe sont trop simples, il est facile de les retrouver à partir de l'empreinte
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

**Solution** : ajouter une valeur supplémentaire au moment du calcul d'empreinte

# Le salage

**Problème** : Deux mots de passe identiques ont la même empreinte

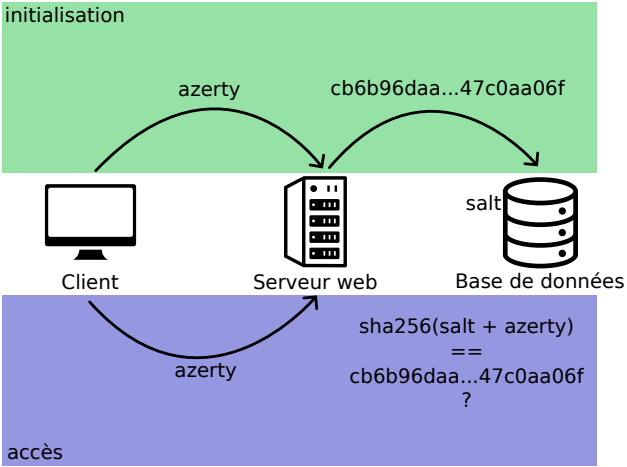
- si les mots de passe sont trop simples, il est facile de les retrouver à partir de l'empreinte
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

**Solution** : ajouter une valeur supplémentaire au moment du calcul d'empreinte

**Résultat** : Une rainbow table ne suffit plus !

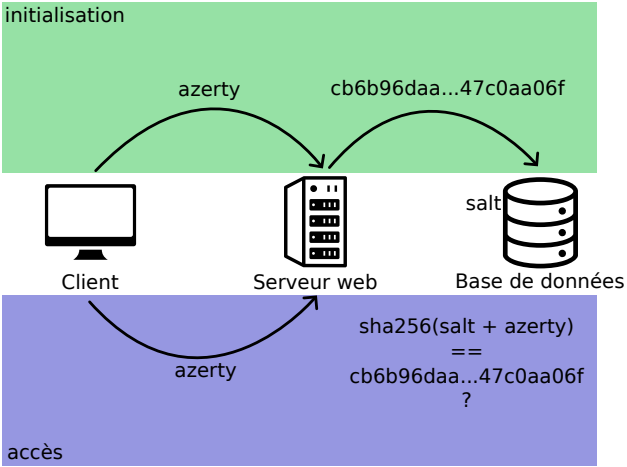
$ echo 'complex_salt_AZERTY' | sha256sum → 0b1b492eb5217e9852a1...0e45154875f20937d3fd3c5
$ echo 'AZERTY' | sha256sum → 194b0cdc29ac7d6f42bc...8365348da649ed2255e1a30

# Stocker le mot de passe hashé et salé dans une base de données ?




- Authentification de l'utilisateur
- Virtuellement impossible de retrouver le mot de passe depuis le condensat

# Stocker le mot de passe hashé et salé dans une base de données ?




+ Authentification de l'utilisateur
+ Virtuellement impossible de retrouver le mot de passe depuis le condensat

- Le sel utilisé devient un secret à protéger
- Avec les ressources suffisantes et le sel il suffit de créer une rainbow table dédiée à notre application.

# Le salage, le vrai

**Problème** : Au sein de notre application deux mots de passe identiques ont la même empreinte

- si les mots de passe sont trop simples, il est possible de les retrouver à partir de l'empreinte et du sel générique
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

# Le salage, le vrai

**Problème** : Au sein de notre application deux mots de passe identiques ont la même empreinte

- si les mots de passe sont trop simples, il est possible de les retrouver à partir de l'empreinte et du sel générique
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

**Solution** : ajouter une valeur aléatoire au moment du calcul d'empreinte

- soit avec une information associée au compte (login, identifiant interne...) : mécanisme déconseillé aujourd'hui
- soit avec une valeur totalement aléatoire, stockée soit ailleurs, soit en début de clé (cas de l'algorithme **bcrypt**)

# Le salage, le vrai

**Problème** : Au sein de notre application deux mots de passe identiques ont la même empreinte

- si les mots de passe sont trop simples, il est possible de les retrouver à partir de l'empreinte et du sel générique
  - calcul de toutes les empreintes pour les mots de passe les plus courants
  - attaque dite par *rainbow table*, ou *table arc-en-ciel*.

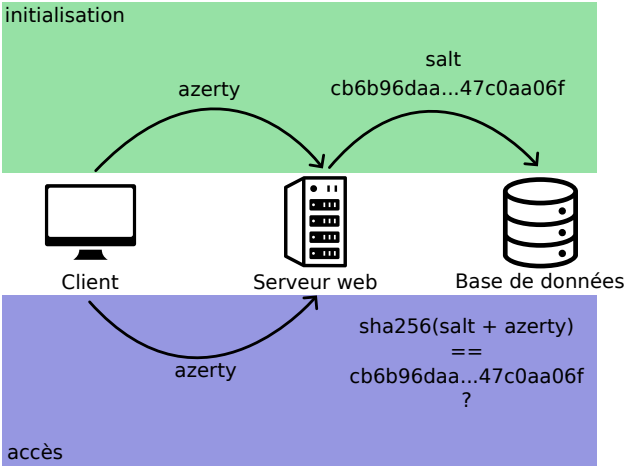
**Solution** : ajouter une valeur aléatoire au moment du calcul d'empreinte

- soit avec une information associée au compte (login, identifiant interne...) : mécanisme déconseillé aujourd'hui
- soit avec une valeur totalement aléatoire, stockée soit ailleurs, soit en début de clé (cas de l'algorithme **bcrypt**)

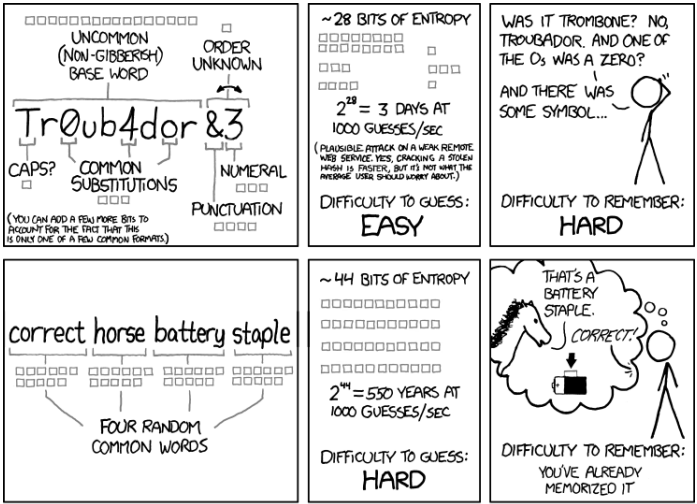
**Résultat** : Deux mots de passe identiques présentent alors deux empreintes différentes

```
$ echo 's1_AZERTY' | sha256sum → 0190f69c589369ced08877...4cb8903fb2419ac3c9610599cd8
$ echo 's2_AZERTY' | sha256sum → f136133406c91ab2be76b8...e305e47547ec1664a6f33554e1a
```

# Stocker le mot de passe hashé et salé dans une base de données !




- Authentification de l'utilisateur
- Virtuellement impossible de retrouver le mot de passe

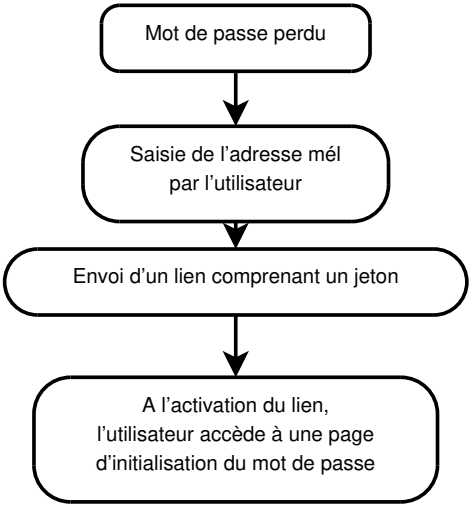



*Source : xkcd.com*

# Recommandations

- L'ANSSI recommande dans son guide (ssi.gouv.fr) :
  - longueur au delà de 9 caractères (en fonction de la sensibilité)
  - vaste jeu de caractère (minuscules, majuscules, chiffres, caractères spéciaux)
  - expiration programmée
  - toujours en fonction de la sensibilité de l'application ;
    - blocage au bout d'un nombre limité d'essais
    - authentification multi-facteurs (sms, Google Auth, etc.)
- stockage en base de données
  - chiffrement non réversible (calcul d'empreinte, comme sha256 ou sha512)
  - adjonction d'un sel variable (minimum 128 bits)
- les contrôles de complexité doivent être réalisés côté navigateur **et** refaits dans le serveur
- verrouillage
  - hard : réinitialisation par un administrateur
  - soft : durée de blocage soit fixe, soit croissante

# Procédure de recouvrement du mot de passe




- jeton généré cryptographiquement
- durée d'expiration courte

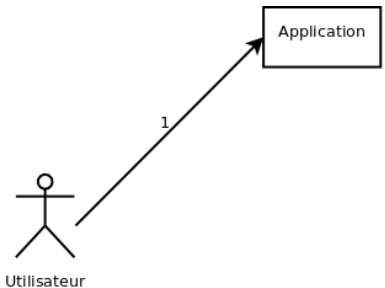
# Identification LDAP

- L'utilisateur est identifié auprès de l'annuaire de l'entreprise
- LDAP : Liteweight Directory Access Protocol
  - largement utilisé dans le monde Linux
  - *Active Directory* est compatible LDAP
- L'authentification est sous-traitée

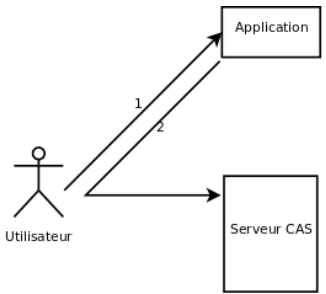
# Identification via un serveur CAS

- **Objectif** : sous-traiter l'authentification à un serveur dédié
  - il s'appuie sur un ou plusieurs annuaires gérés par l'entreprise ou ses partenaires
    - cas de la fédération Renater, regroupant les universités et organismes de recherche publique en France
    - liens avec les fédérations étrangères...
  - l'application n'a plus à gérer l'identification des utilisateurs
  - elle ne peut pas connaître le mot de passe
  - permet d'utiliser des mécanismes complexes (cartes à puces *p. e.*) gérés par le serveur dédié
- **Inconvénient** : tous les utilisateurs de l'application doivent être connus dans l'annuaire central
  - recours à deux mécanismes d'identification concurrents dans l'application (bouton CAS ou login/mdp)

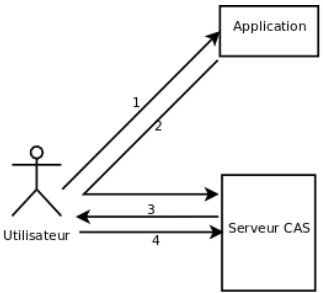
# Principe de fonctionnement



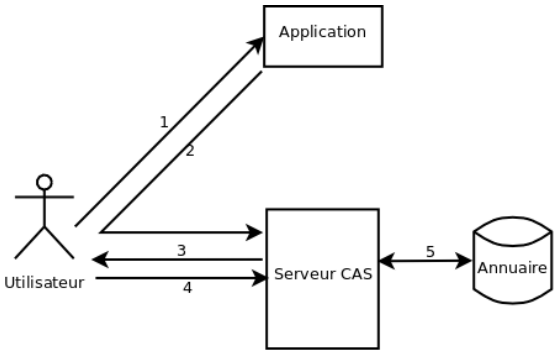
# Principe de fonctionnement



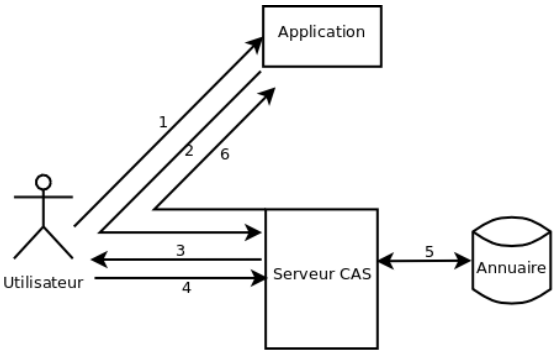
# Principe de fonctionnement



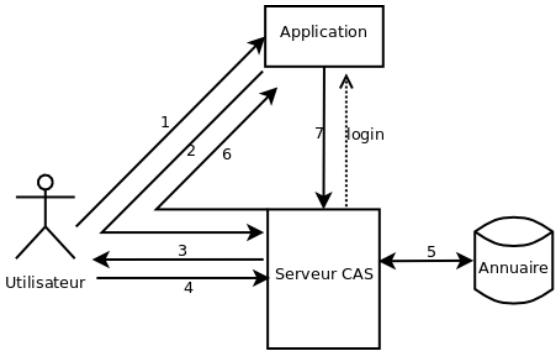
# Principe de fonctionnement



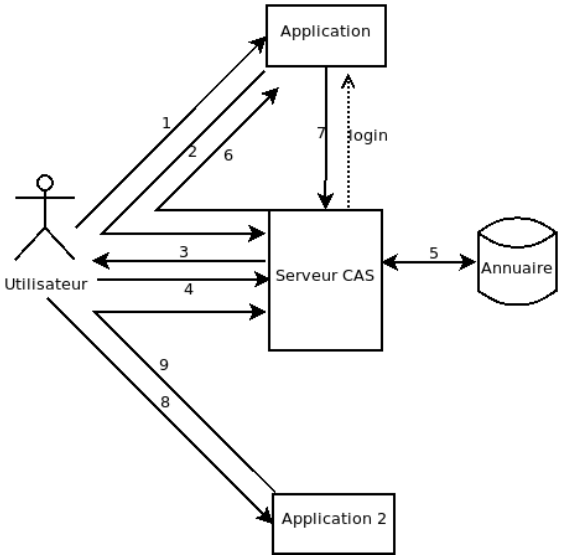
# Principe de fonctionnement



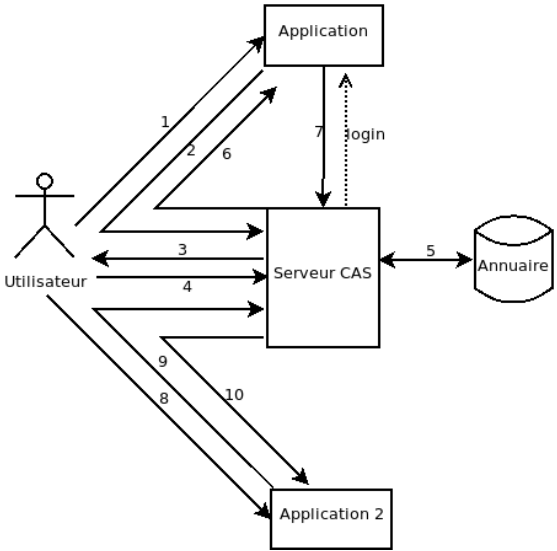
# Principe de fonctionnement



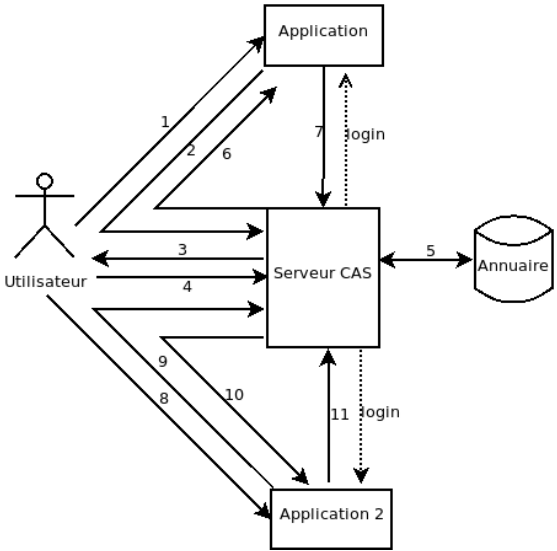
# Principe de fonctionnement



# Principe de fonctionnement



# Principe de fonctionnement



# Principe de fonctionnement

1. l'utilisateur veut se connecter à l'application
2. l'application ne le connait pas, et le renvoie vers le CAS
3. le CAS ne le connaît pas, et engage le dialogue
4. l'utilisateur s'identifie
5. le serveur CAS vérifie l'identification auprès de l'annuaire
6. le serveur CAS le renvoie vers l'application d'origine, et fournit un numéro de ticket
7. l'application, avec le ticket, récupère le login
8. l'utilisateur veut se connecter à une seconde application
9. l'application ne le connaît pas, et le renvoie vers le serveur CAS
10. le serveur CAS connaît l'utilisateur, et le renvoie vers l'appli 2, avec un nouveau ticket
11. l'appli 2 récupère le login auprès du CAS

C'est le principe du SSO (*Single Sign One*) : l'utilisateur ne s'identifie qu'une fois, en un point

# Identification via jeton - le protocole JWT

- principe : une fois identifié une première fois, le login est stocké dans un jeton cryptographique
  - chiffré par le serveur d'authentification
  - transmis au navigateur
  - à chaque fois que nécessaire, le jeton est présenté au serveur de l'application
- le protocole JWT (*Json Web Token*) normalise son contenu
- forme : xxx.yyy.zzz
  - xxx : protocole utilisé

    ```
    header = '{"alg":"HS256","typ":"JWT"}'
    ```

  - yyy : contenu du jeton

    ```
    payload = '{"loggedInAs":"admin","iat":1422779638}'
    ```

  - zzz : signature : résultat du chiffrement de xxx.yyy

# Identification via jeton - le protocole JWT

- Pour garantir l'authentification, il vaut mieux recourir au chiffrement RS256 :
  - chiffrement avec la clé privée
  - déchiffrement (par tous) avec la clé publique pour garantir l'origine
- Variables principales normalisées utilisables dans le contenu :

| code | Nom complet | Description |
|---|---|---|
| iss | Issuer | Nom du fournisseur du jeton |
| sub | Subject | Utilisation principale |
| aud | Audience | À qui s'adresse le jeton |
| exp | Expiration time | Timestamp d'expiration |
| nbf | Not before | Timestamp de date mini |
| iat | Issued at | Timestamp de génération |
| jti | JWT ID | Identifiant unique |

- Aucune n'est techniquement obligatoire

# Le protocole Oauth2

- Créé pour récupérer des informations auprès d'un serveur tiers :
  - l'utilisateur s'identifie auprès du serveur tiers et autorise ou non l'application à récupérer ses informations
  - extension du CAS pour pouvoir récupérer des données, et pas seulement gérer l'identification

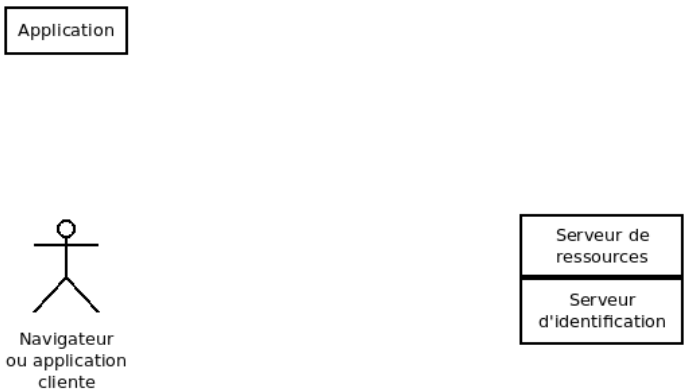
# Le protocole Oauth2

- Créé pour récupérer des informations auprès d'un serveur tiers :
  - l'utilisateur s'identifie auprès du serveur tiers et autorise ou non l'application à récupérer ses informations
  - extension du CAS pour pouvoir récupérer des données, et pas seulement gérer l'identification
- Même principe de base que le CAS :
  - le serveur redirige le navigateur vers le serveur d'authentification
  - le navigateur transmet au serveur la réponse du serveur d'authentification
  - les informations sont obtenues directement par le serveur auprès du serveur distant

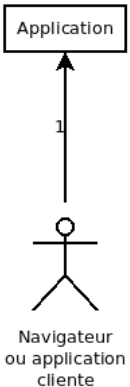
# Le protocole Oauth2

- Créé pour récupérer des informations auprès d'un serveur tiers :
  - l'utilisateur s'identifie auprès du serveur tiers et autorise ou non l'application à récupérer ses informations
  - extension du CAS pour pouvoir récupérer des données, et pas seulement gérer l'identification
- Même principe de base que le CAS :
  - le serveur redirige le navigateur vers le serveur d'authentification
  - le navigateur transmet au serveur la réponse du serveur d'authentification
  - les informations sont obtenues directement par le serveur auprès du serveur distant
- Au préalable, l'application est enregistrée auprès du serveur d'identification
  - elle fournit notamment une adresse de retour
  - le serveur d'identification fournit :
    - un identifiant (*appli_id*), public
    - un *secret*, qui doit rester privé

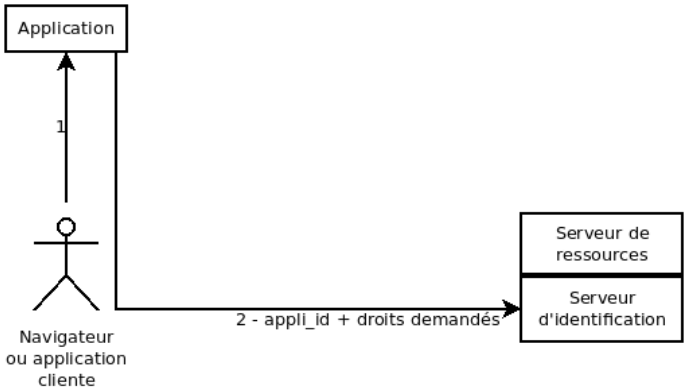
# Principe de fonctionnement



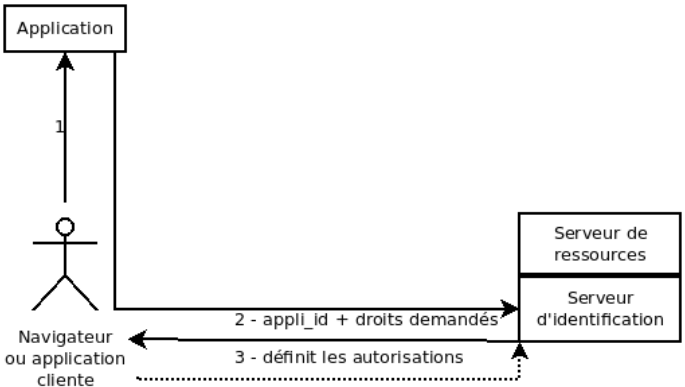
# Principe de fonctionnement



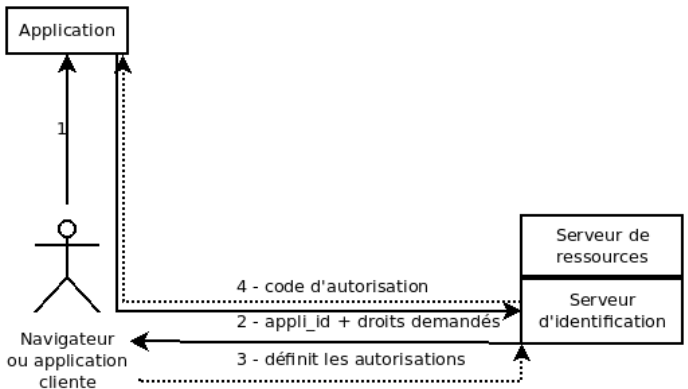

# Principe de fonctionnement



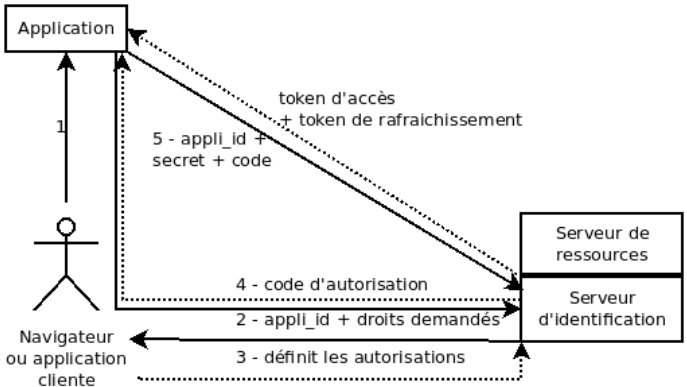
# Principe de fonctionnement



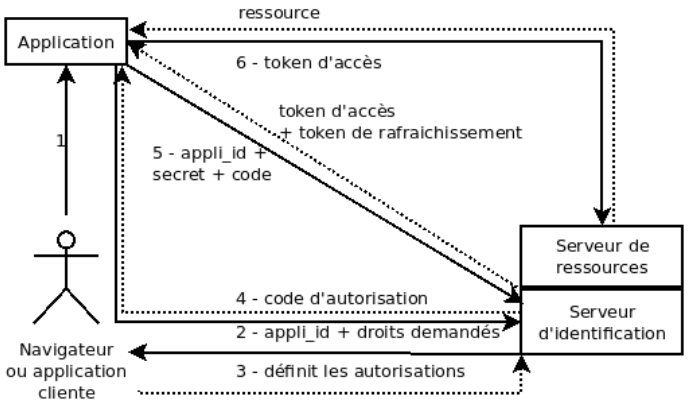
# Principe de fonctionnement



# Principe de fonctionnement



# Principe de fonctionnement



# Principe de fonctionnement

1. l'utilisateur se connecte à l'application, ou veut lui faire exécuter des actions
2. l'application redirige l'utilisateur vers le serveur d'identification, en fournissant son *appli_id*
3. le serveur d'identification instaure un dialogue avec l'utilisateur, et récupère son assentiment pour que l'application puisse utiliser ses ressources
4. le serveur d'identification renvoie le navigateur vers le serveur, en fournissant un *code* d'autorisation
5. l'application contacte le serveur d'identification et lui fournit son *appli_id*, son *secret* et le *code* d'autorisation. Elle récupère un token d'accès et un token de rafraichissement, pour éviter à l'utilisateur de redonner ses droits d'accès à l'expiration du token d'accès
6. l'application présente le token d'accès au serveur de ressources pour récupérer les ressources qui ont été autorisées

# OpenID Connect

*OpenID Connect* est une surcouche de Oauth2 pour fournir l'authentification de l'utilisateur.

- Elle fonctionne sous la forme de services REST et fournit l'information sous forme de jetons JWT
- Le jeton contient un certain nombre de champs, dont l'identifiant, spécifique au serveur d'authentification

# Quand utiliser OpenID Connect ?

- Avantages :
  - l'utilisateur peut se connecter à partir de services tiers
  - Plus complet que le protocole CAS
- Inconvénients :
  - il faut gérer des identifiants multiples pour un même utilisateur
  - selon le serveur, il peut disposer d'adresses mails différentes
  - l'application doit être enregistrée auprès de chaque fournisseur d'identité
- Cas d'usage :
  - dans les applications « grand public », par exemple www.overleaf.com
    - application collaborative d'écriture de documents en LaTeX
    - propose l'authentification via Google ou ORCID
  - le choix des fournisseurs d'identité doit être cohérent avec la cible visée
- Tous les fournisseurs d'identité ne proposent pas OpenID Connect
- Plus complexe qu'un serveur CAS pour une simple application interne

# La double identification avec TOTP

- Objectif : doubler l'authentification par mot de passe avec un dispositif physique détenu par l'utilisateur
- Principe du TOTP (*Time-based One-time Password*) :
  - une clé secrète est générée par l'application, et associée au compte de l'utilisateur
  - la clé secrète est copiée dans un terminal appartenant à l'utilisateur :
    - smartphone
    - carte à puce, dispositif dédié...
  - au moment de l'authentification, une clé temporaire est générée :
    - une durée de validité en général de 30 secondes
    - une longueur de 6 chiffres
- Seul le détenteur du dispositif pourra se connecter
- Pour une explication de l'algorithme : www.ionos.fr/digitalguide/serveur/securite/totp

# Quand utiliser TOTP ?

- Quand utiliser TOTP ?
  - pour les applications critiques
  - pour les administrateurs des applications
  - pour accéder à des fonctions nécessitant des droits élevés
- Limites :
  - en cas de perte ou de réinitialisation du terminal, l'utilisateur ne peut plus se connecter
    - prévoir un mécanisme de secours :
    - réinitialisation par un administrateur (applications avec un faible volume de personnes concernées)
    - codes de secours pour réinitialiser la fonctionnalité (utilisé par Google, Github, Gitlab, ...)

# Crédits

Document inspiré du cours « Identification des utilisateurs »de E. Quinton, eric.quinton@protonmail.com