SRIV - Sécurité réseau

Authentification sur le WEB

Fabien Rico (fabien.rico@univ-lyon1.fr)
Vincent Hurtevent

Univ. Claude Bernard Lyon 1

Authentification à clef publique

- Elle permet à un acteur d'être reconnu par n'importe quel système.
- Elle nécessite de transmettre la clef publique sans modification.
- Il faut que la clef soit signée par un tiers de confiance.

Le couple clef publique/privé permet une authentification efficace, mais il faut un moyen de distribuer les clefs publiques de manières sûre. Si un attaquant peut remplacer la clef publique d'un serveur par la sienne, il peut se faire passer pour lui.

Certificat x509

C'est une carte d'identité numérique qui contient au moins :

- des informations sur le titulaire du certificat (DN, mail, ...) ;
- une période de validité ;
- la clef publique du titulaire ;
- l'autorité de certificat ;
- la signature de cette autorité.

À partir du certificat et de la clef de l'autorité il est possible de vérifier ce dernier.


Pourquoi limiter la validité dans le temps


Validité d'un certificat

En plus de sa période de validité, le certificat peut être invalidé :

- dès qu'un champs est modifié ;
- parce que la clef est compromise ;
- parce que service est remplacé ;
- ...

Tout les cas d'invalidation ne sont pas directement vérifiables, l'autorité maintient donc une *liste de révocation* des certificats émis.

Chaine de certification

Pour que cela fonctionne, la clef publique de l'autorité doit être déjà présente dans le magasin de clefs de celui qui vérifie.

- On utilise une chaine de certification :
  - une autorité locale certifie le serveur ;
  - son certificat lui même signé par une autorité ;
  - ...
  - le certificat racine (autosigné) est distribué autrement.
- Un certains nombre de certificats racines sont présent dans tous les systèmes.

Le nombre de certificats racines est assez important, beaucoup sont détenus par des entreprises privées qui vendent la certification.

Limite du système

- Très grande centralisation (à comparer avec les DNS).
- Difficulté à automatiser la création de certificats.
- Pas de différenciation entre les risques :
  - certificat d'une banque ;
  - certificat d'un service interne à une entreprise ;
  - certificat d'un site web personnel.
- Beaucoup d'autorités.
- Peu d'explication auprès du public (que signifie le petit icon de sécurité ?).

Le résultat est qu'on voit beaucoup de site sans certificat, ou de message peu clair sur les risques encouru.

Que ce passe-t-il si la date de votre ordinateur est erronée ?

Let's encrypt

Un système pour automatiser la génération de certificat

- En version beta depuis décembre 2015.
- Le service est ouvert et gratuit, mais nécessite une machine qui se connecte à internet.
- Fonctionnement :
  - Un client demande la génération de certificats de durée de vie courte qu'il place lui même sur les serveurs.
  - Un ensemble de robots interrogent régulièrement le client pour vérifier qu'il n'y a pas de vol d'identité.
  - Les vérifications sont effectuées depuis plusieurs endroit différents afin d'éviter les attaques via le DNS (empoisonnement de cache par ex.).
- Le titulaire d'un domaine DNS peut générer un premier certificat directement.
- Si un certificat à déjà été délivré, on peut ne peut en générer de nouveau qu'à partir d'un certificat valide.

Authentification centralisée

L'utilisation d'annuaire permet de centraliser l'authentification. Avec votre mot de passe vous pouvez :

- vous connecter sur les ordinateurs gérés par le cri (qqsoit le système) ;
- vous connecter sur le vpn (géré par le cri) ;
- accéder au réseau wifi ;
- accéder à l'intranet (géré par le service web du cri) ;
- regarder vos notes sur tomuss (géré par T. Excofier et E. Coquery) ;
- consulter les pages spiral (géré par icap) ;
- accéder aux revues en ligne (géré par ? ? ?)
- s'authentifier sur le serveur apache d'un cours
- ...

N'y a-t'il pas un problème

Single Sign On

- Centraliser l'authentification et pas seulement les informations :
  - Ne pas transmettre de mot de passe aux différents serveurs.
  - Authentification par un tiers de confiance (ex : banque).
  - Preuve d'authentification :
    - ★ donnée par l'autorité de confiance,
    - ★ dépendant du client.
- Signature une seule fois :
  - Ne pas redemander les informations d'authentification.
  - Ne pas permettre à un espion de rejouer l'authentification :
    - ★ auprès d'un autre serveur ;
    - ★ une seconde fois auprès du même serveur.

Single Sign On

Principe du passeport

- Une autorité est capable de vous authentifier.
- Elle est reconnue par plusieurs services.
- L'autorité
  - reçoit les informations de connexion ;
  - vous accrédite auprès du service ;
  - en général ne demande qu'une fois les informations.

Exemple :

- Kerberos : authentification dans le réseaux.
- CAS pour le web.
- Fédérations d'identité : *Shibboleth* ou Athens

Kerberos [Sta02]

- Développé à partir du projet Athéna au MIT.
- Version actuelle 5
- Protocole d'authentification par défaut pour windows, possible pour unix (PAM), NFS, openssh, samba, ...
- Utilise des échanges chiffrés par clefs symétriques (initialement DES)
- Architecture :
  - client ;
  - serveur demandant l'authentification ;
  - serveur d'authentification (qui vérifie l'identité) ;
  - serveur d'octroi de ticket TGS (qui crée un ticket utilisable).

Kerberos

Serveur

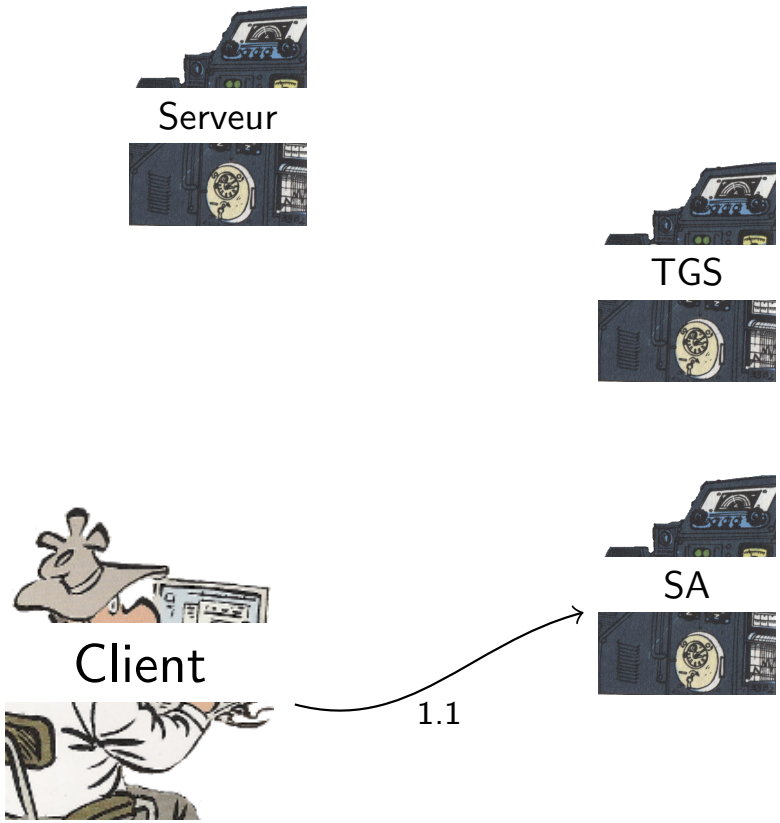
TGS

Client

SA

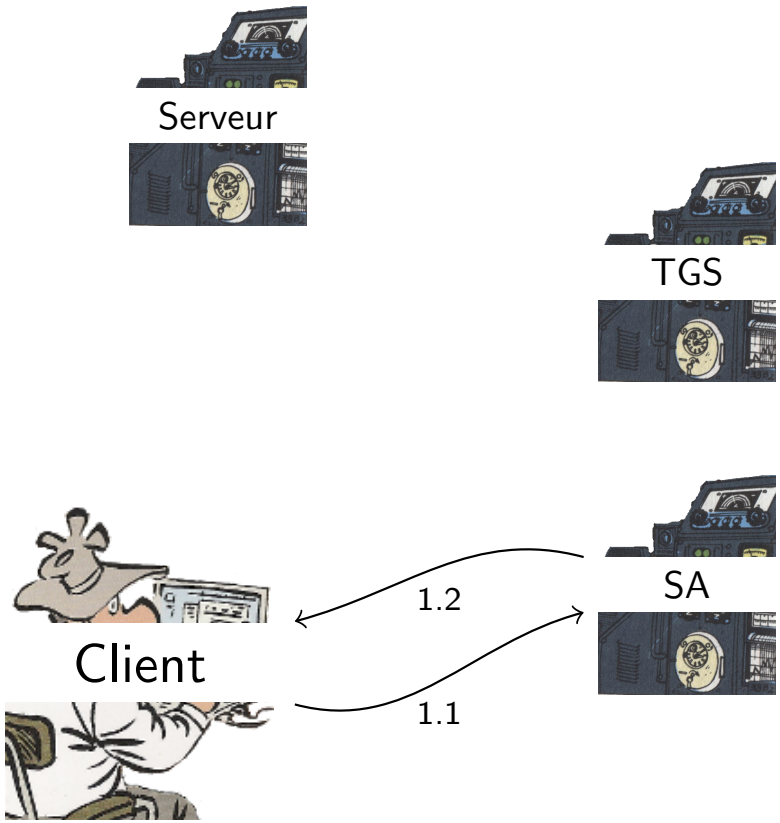
1. Authentification (1 fois par session)

Kerberos



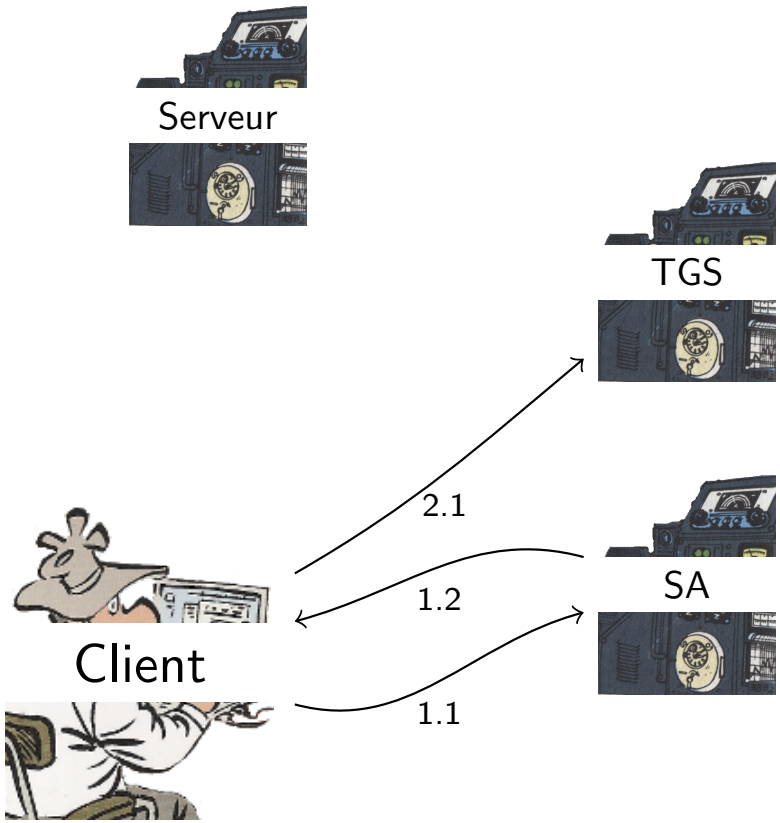
1. Authentification (1 fois par session)
   1. le client demande authentification ;

Kerberos



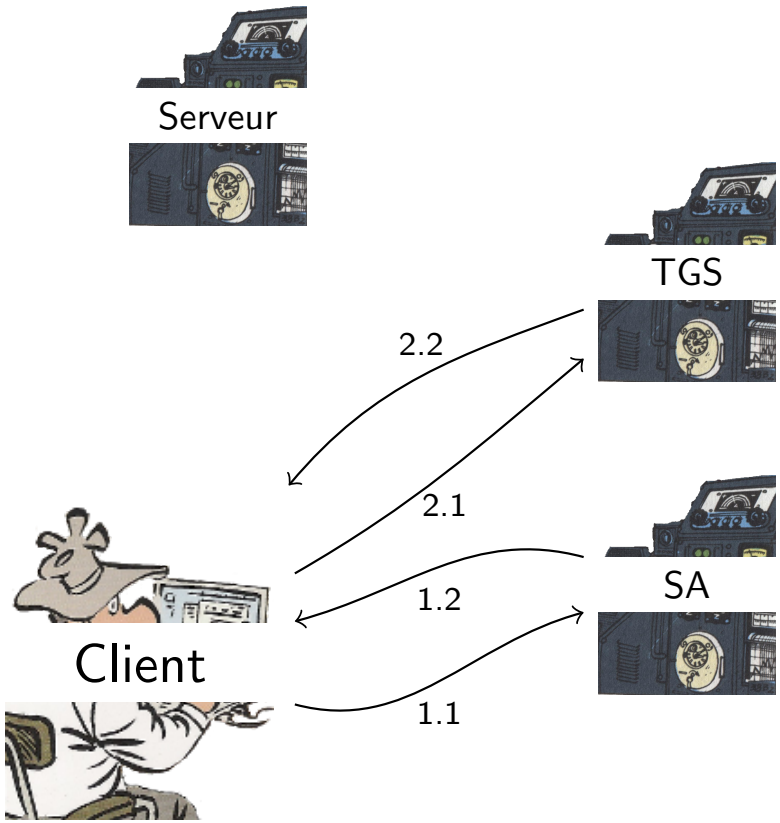
1. Authentification (1 fois par session)
   1. le client demande authentification ;
   2. le SA fourni un ticket TGS.

Kerberos



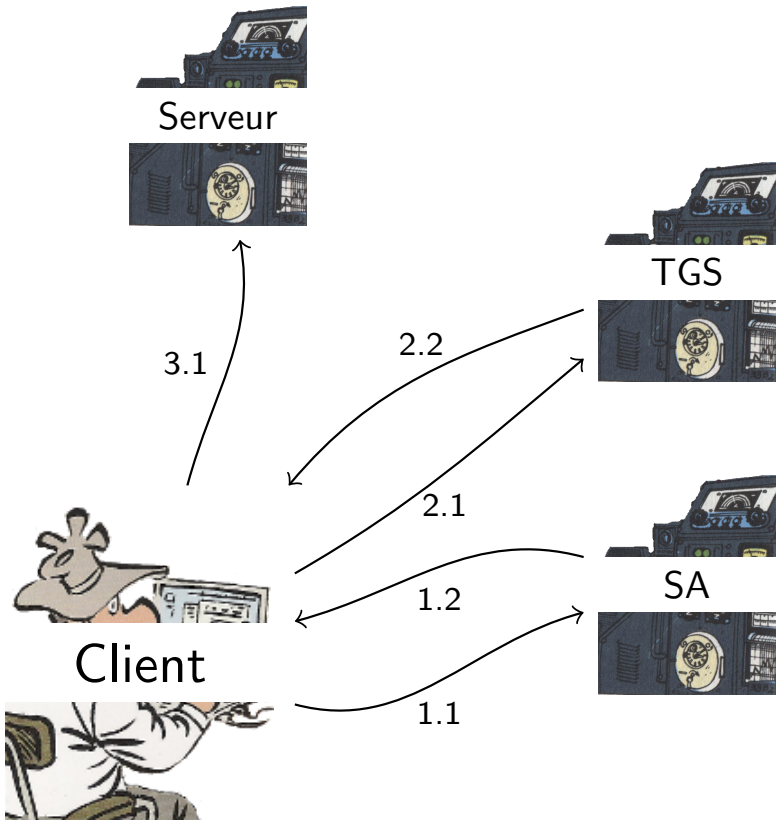
1. Authentification (1 fois par session)
   1. le client demande authentification ;
   2. le SA fourni un ticket TGS.
2. Récupération d'un ticket de service (1 fois pour chaque service)
   1. le client envoie demande d'accès avec ticket TGS et preuve d'identité ;

Kerberos



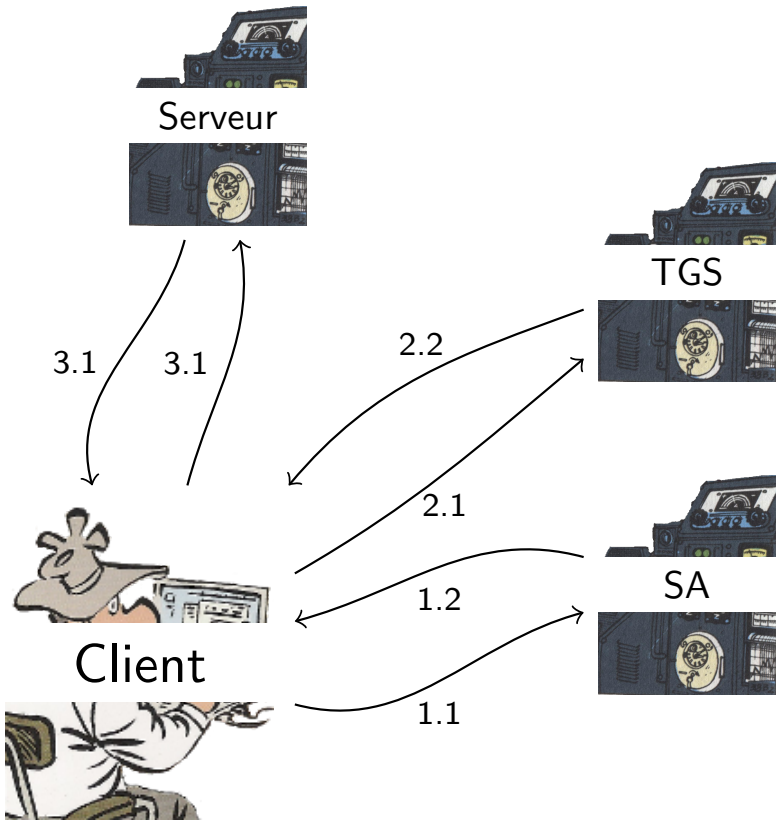
1. Authentification (1 fois par session)
   1. le client demande authentification ;
   2. le SA fourni un ticket TGS.
2. Récupération d'un ticket de service (1 fois pour chaque service)
   1. le client envoie demande d'accès avec ticket TGS et preuve d'identité ;
   2. le TGS fourni un ticket serveur à utilisation unique.

Kerberos



1. Authentification (1 fois par session)
   1. le client demande authentification ;
   2. le SA fourni un ticket TGS.
2. Récupération d'un ticket de service (1 fois pour chaque service)
   1. le client envoie demande d'accès avec ticket TGS et preuve d'identité ;
   2. le TGS fourni un ticket serveur à utilisation unique.
3. Accès au service (1 fois pour chaque service) :
   1. le client envoie le ticket serveur et une preuve d'identité ;

Kerberos



1. Authentification (1 fois par session)
   1. le client demande authentification ;
   2. le SA fourni un ticket TGS.
2. Récupération d'un ticket de service (1 fois pour chaque service)
   1. le client envoie demande d'accès avec ticket TGS et preuve d'identité ;
   2. le TGS fourni un ticket serveur à utilisation unique.
3. Accès au service (1 fois pour chaque service) :
   1. le client envoie le ticket serveur et une preuve d'identité ;
   2. le serveur accorde l'accès

CAS : Central Authentication Service

- Développé à Yale.
- Permet à plusieurs sites web de demander l'authentification à un unique serveur.
- Répandu dans le domaine universitaire
  - intégré à u-portal ;
  - nombreuses bibliothèques (php, java, ASP, .net, . . .)

CAS : Central Authentication Service

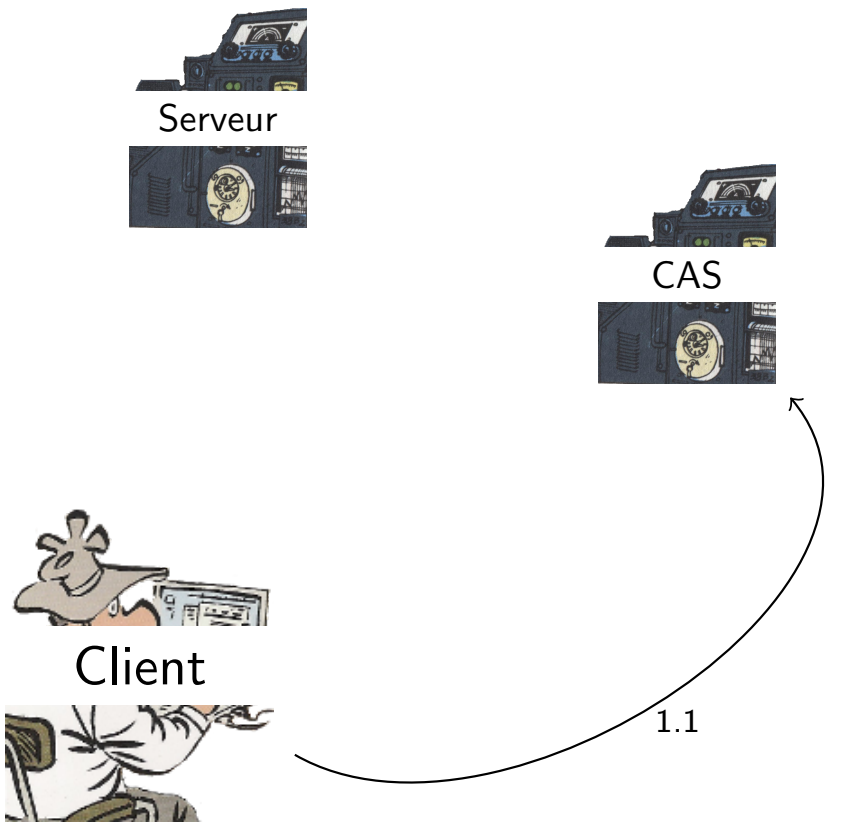
Serveur

CAS

Client

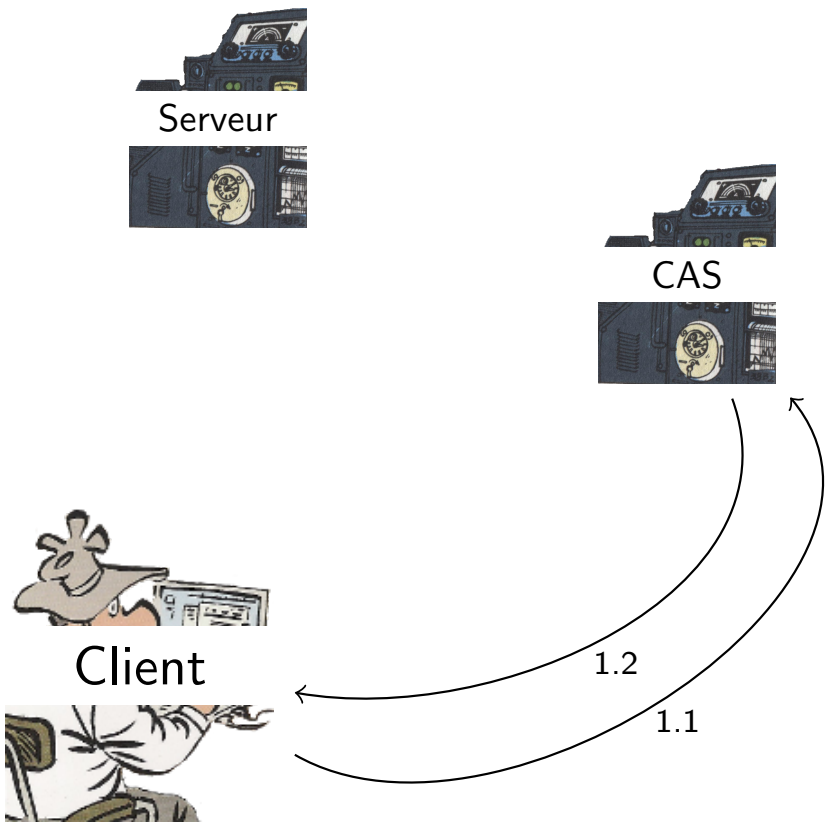
1. Authentification (1 fois par session, généralement au premier accès)

CAS : Central Authentication Service



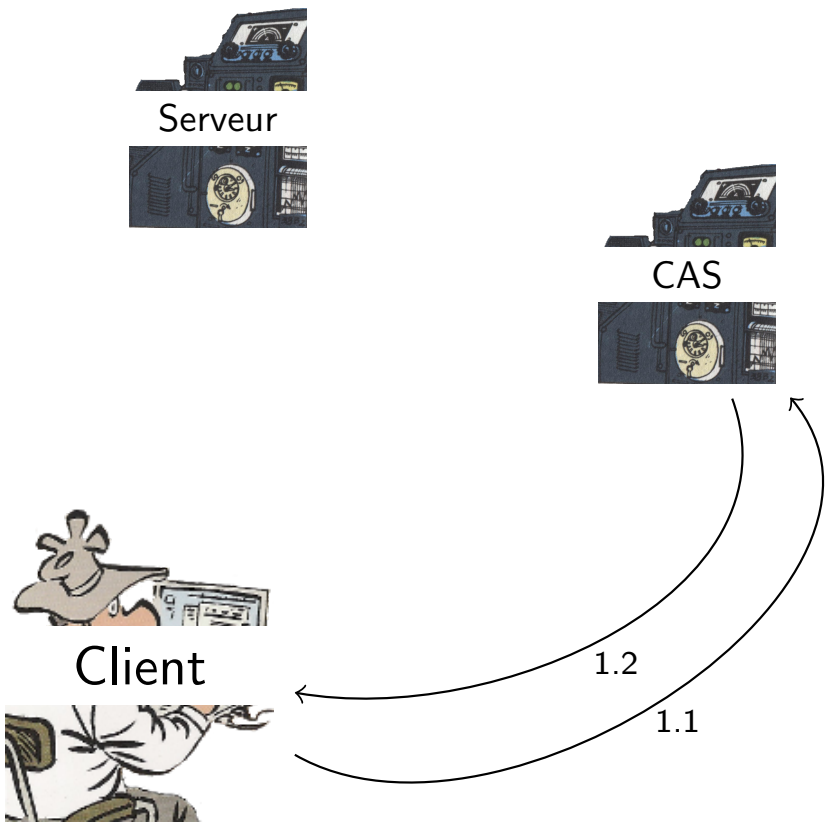
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;

CAS : Central Authentication Service



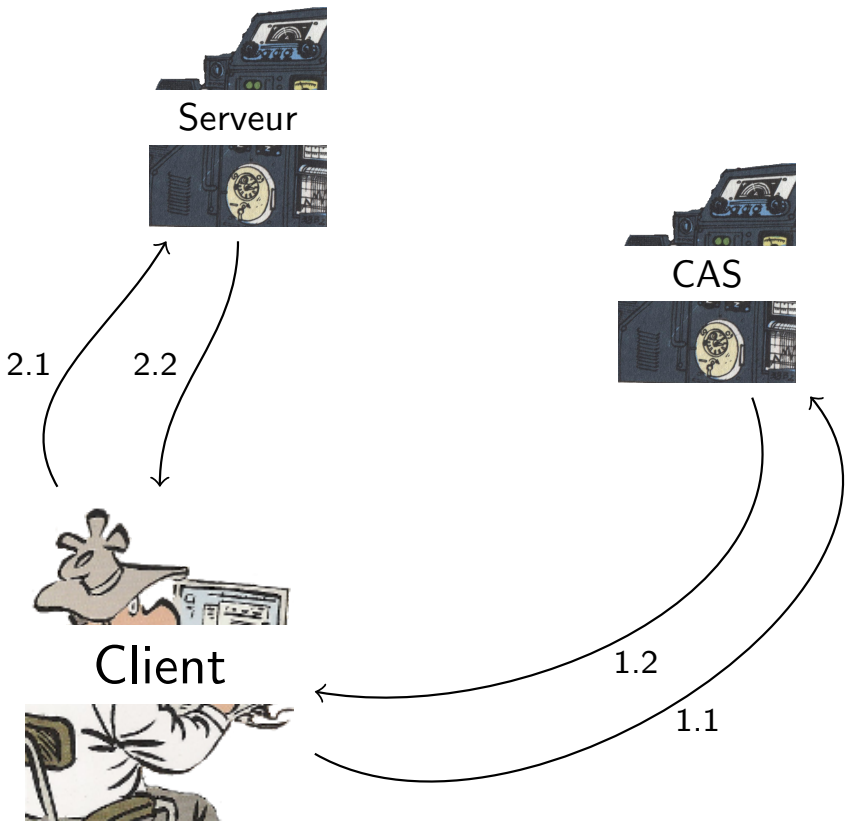
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).

CAS : Central Authentication Service



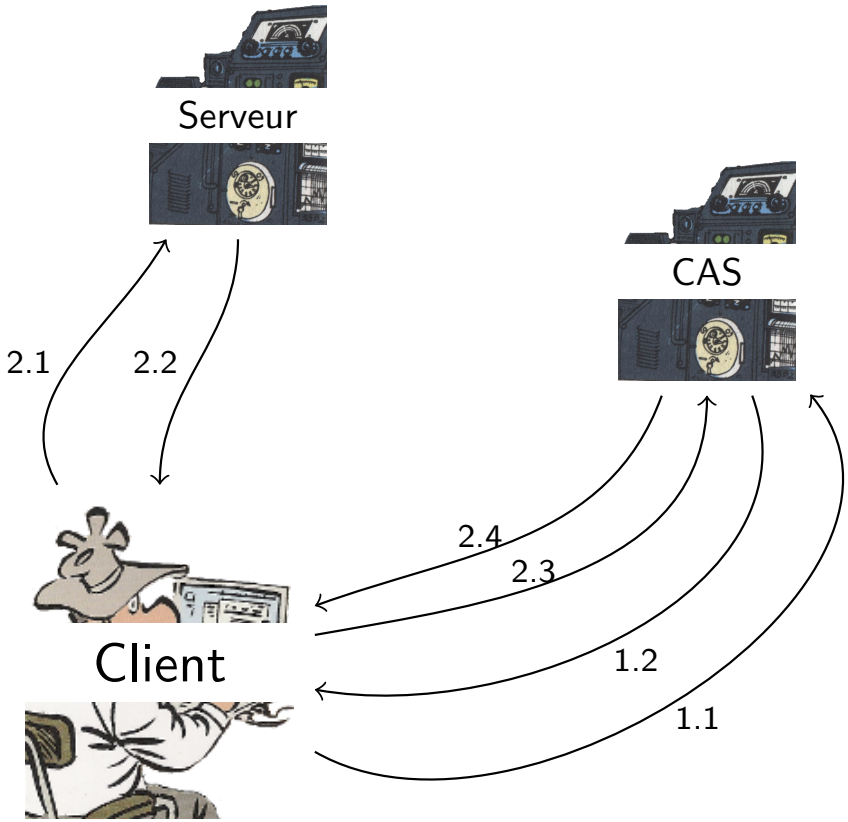
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)

CAS : Central Authentication Service



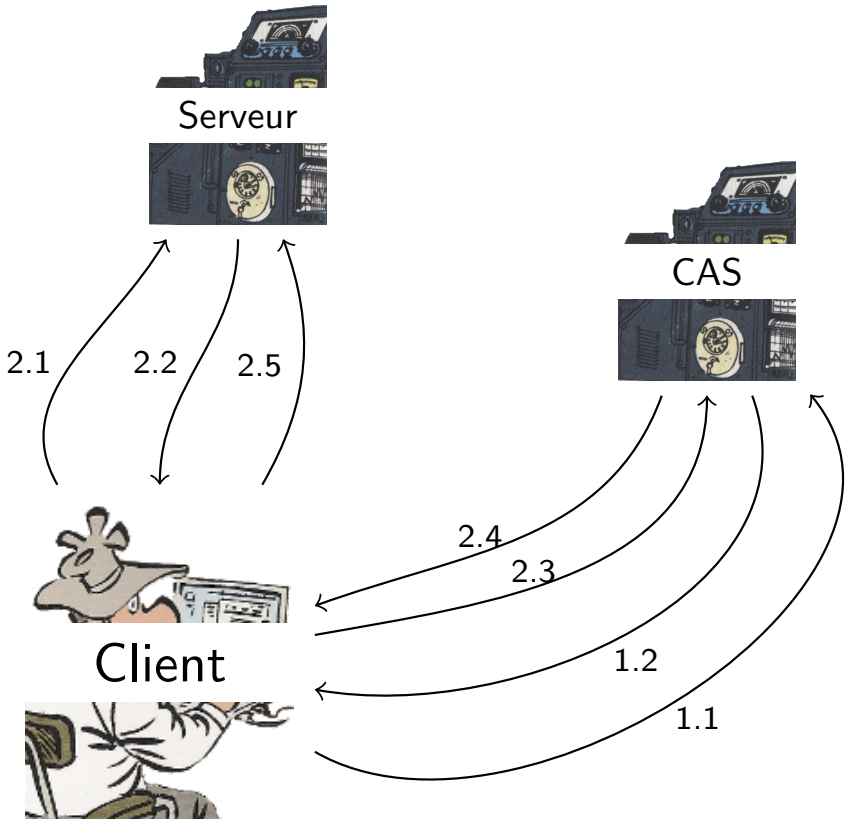
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)
   1. le client se connecte au serveur ;
   2. le serveur renvoie le client sur le CAS ;

CAS : Central Authentication Service



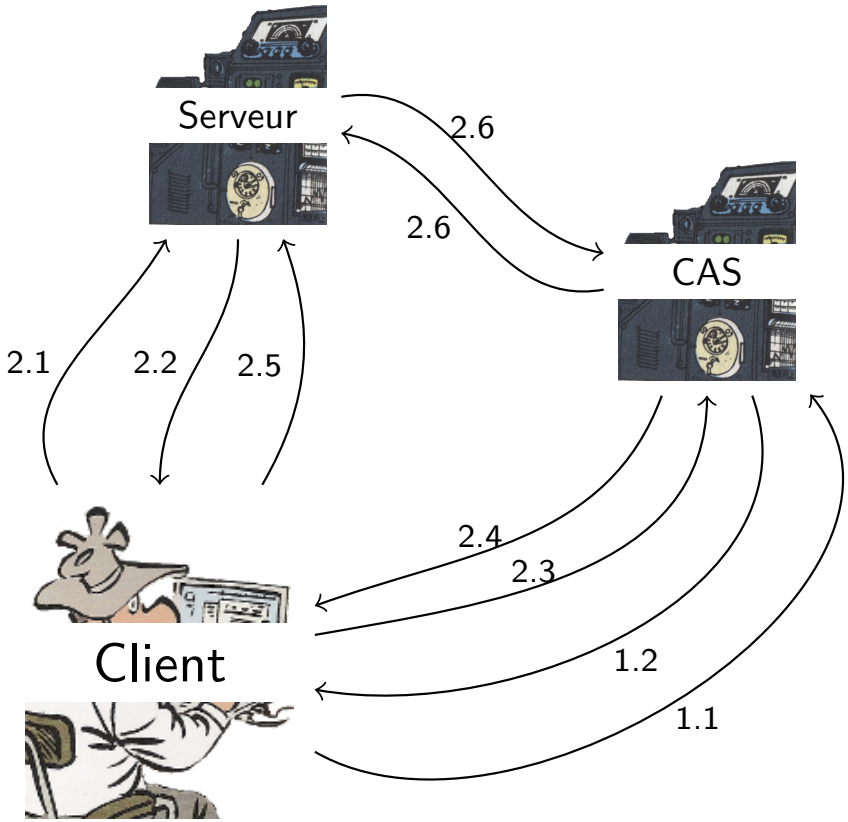
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)
   1. le client se connecte au serveur ;
   2. le serveur renvoie le client sur le CAS ;
   3. le client demande un ticket pour le serveur ;
   4. le CAS renvoie un ticket de service (ST) et redirige le client vers le serveur ;

CAS : Central Authentication Service



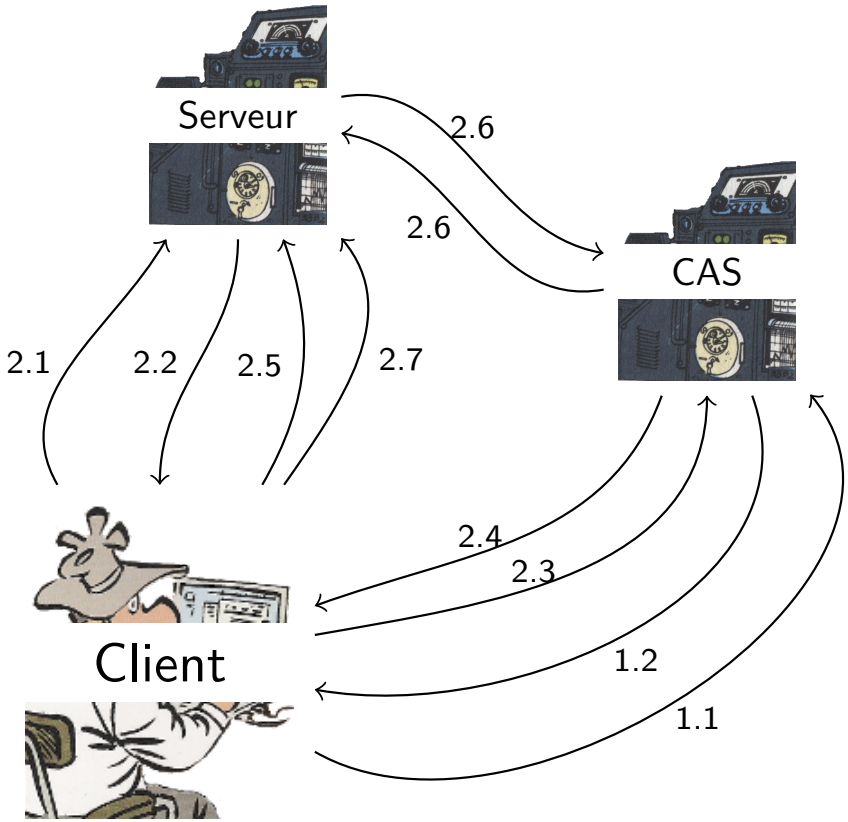
1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)
   1. le client se connecte au serveur ;
   2. le serveur renvoie le client sur le CAS ;
   3. le client demande un ticket pour le serveur ;
   4. le CAS renvoie un ticket de service (ST) et redirige le client vers le serveur ;
   5. le client présente son ticket ;

CAS : Central Authentication Service



1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)
   1. le client se connecte au serveur ;
   2. le serveur renvoie le client sur le CAS ;
   3. le client demande un ticket pour le serveur ;
   4. le CAS renvoie un ticket de service (ST) et redirige le client vers le serveur ;
   5. le client présente son ticket ;
   6. le serveur verifie le ticket ;

CAS : Central Authentication Service



1. Authentification (1 fois par session, généralement au premier accès)
   1. le client demande authentification ;
   2. le SA fourni un cookie de session (TGC Ticket-Granting Cookie).
2. Authentification auprès d'un serveur (à chaque accès)
   1. le client se connecte au serveur ;
   2. le serveur renvoie le client sur le CAS ;
   3. le client demande un ticket pour le serveur ;
   4. le CAS renvoie un ticket de service (ST) et redirige le client vers le serveur ;
   5. le client présente son ticket ;
   6. le serveur verifie le ticket ;
   7. le serveur accorde le service.

Exemple client phpCAS

- initialiser le client phpCAS::client
- Authentifier l'utilisateur :
  - phpCAS::forceAuthentication
  - phpCAS::checkAuthentication
- Déloguer phpCAS : :logout() ;

Exemple

Exemple (Authentification simple)

```
include_once('CAS.php');
phpCAS::client(CAS_VERSION_2_0,'cas.univ-lyon1.fr',443,'/cas/');
phpCAS::setNoCasServerValidation();
if (isset($_GET['logout'])) {
  phpCAS::logout();
}
if (isset($_GET['login'])) {
  phpCAS::forceAuthentication();
}
$auth = phpCAS::checkAuthentication();
if ($auth) {
  echo "L'utilisateur est ".phpCAS::getUser()."<Br>";
  echo "<a href='?logout'>se d&eacute;connecter</a><Br>";
} else {
  echo "L'utilisateur n'est pas authentifi&eacute;<Br>";
  echo "<a href='?login'>se connecter</a><Br>";
}
```

Problèmes

- Mot de passe unique :
  - une seule vérification,
  - accès total.
- Mais il faut prendre en considération le comportement des utilisateurs.
- De plus seule solution à certains problèmes :
  - décentralisation des informations,
  - accès distant,
  - partage entre entités.

Transfert d'information utilisateurs

Des applications peuvent avoir besoin de plus d'informations (nom, prénom, mail, status,...) :

- pour pré-remplir des formulaires
- pour traiter différemment certains utilisateurs (personnels/usagés)

Mais on ne peux pas toujours leur donner accès aux serveurs d'informations internes.
Il faut donc un moyen pour que les informations soient échangées de manière sécurisées sans être diffusée.

Confiance

Pour éviter la diffusion, il y a 2 approches :

Définition (Approche fédérative)

Des institutions se coordonnent pour établir un cercle de confiance au sein duquel des informations utilisateurs pourront être transmises
Ex : SAML, Liberty Alliance : réunit des fournisseurs d'identité et des fournisseurs de services identifiés et maintient le cercle de confiance

Définition (Approche centrée utilisateur)

Ou "user-centric", il n'y a pas de cercle de confiance définit, c'est à l'utilisateur de déterminer s'il autorise un service donné à accéder à ses données personnelles auprès de son fournisseur d'identité.
Ex : OpenID

Vocabulaire

Définition (Fournisseur d'identité - Identity Provider - IDP)

Entité spécialisée dans la fourniture d'information relative à l'utilisateur. C'est celui dont les serveur contiennent votre compte et les informations.

Définition (Fournisseur de Service - *Service Provider* - SP)

Entité spécialisée dans la fourniture de service aux utilisateurs (Site web, Application en ligne, etc). C'est le service web que vous souhaité utiliser.

Définition (Assertion ou *Token*)

Message structuré, signé, éventuellement chiffré, échangé entre IDP et SP.

SAML

Définition (Security Assertion Markup Language)

C'est un standard d'échange d'information via XML. Il définit la manière dont les différentes parties, l'IDP et le SP vont mener leurs échanges et comment les messages doivent être construits

- SAML 1.0 : : norme par OASIS en 2002
- SAML 1.1 : mise à jour en 2003
- SAML 2.0 : standard 2005, construit sur SAML 1.1, ID-FF (Liberty Alliance, Shibboleth)

Shibboleth est une implémentation de SAML, il est utilisé dans le monde académique pour assurer l'authentification et le transfert d'informations sur les utilisateur.

Métadonnées

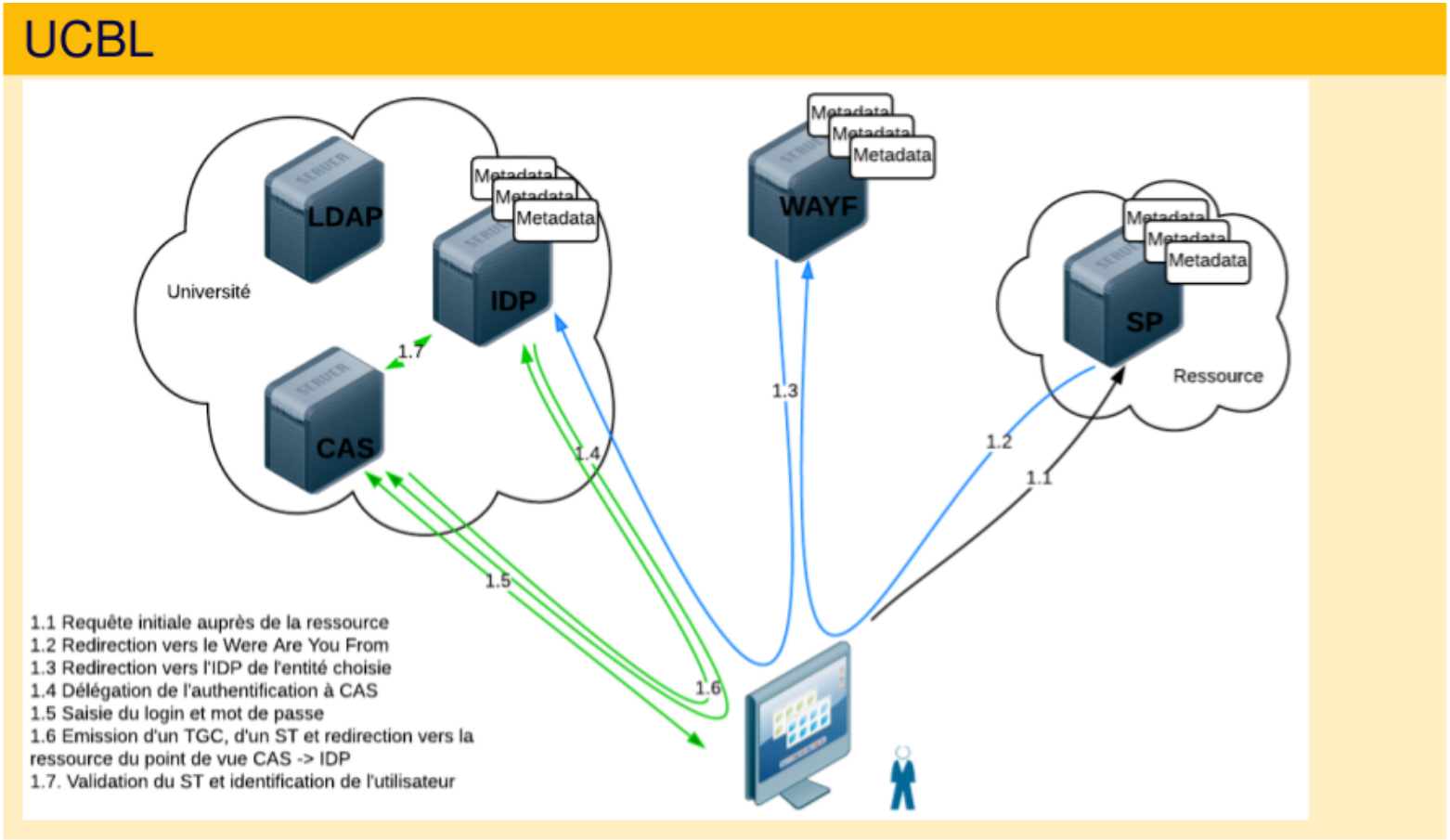
Dans une fédération, toute partie possède ses propres métadonnées, c'est à dire une fiche d'identité la concernant.
Les métadonnées définissent :

- Un nom : EntityID
- Des points d'accès (URL de service)
- Un certificat de clé publique pour signer et/ou chiffrer les assertions SAML

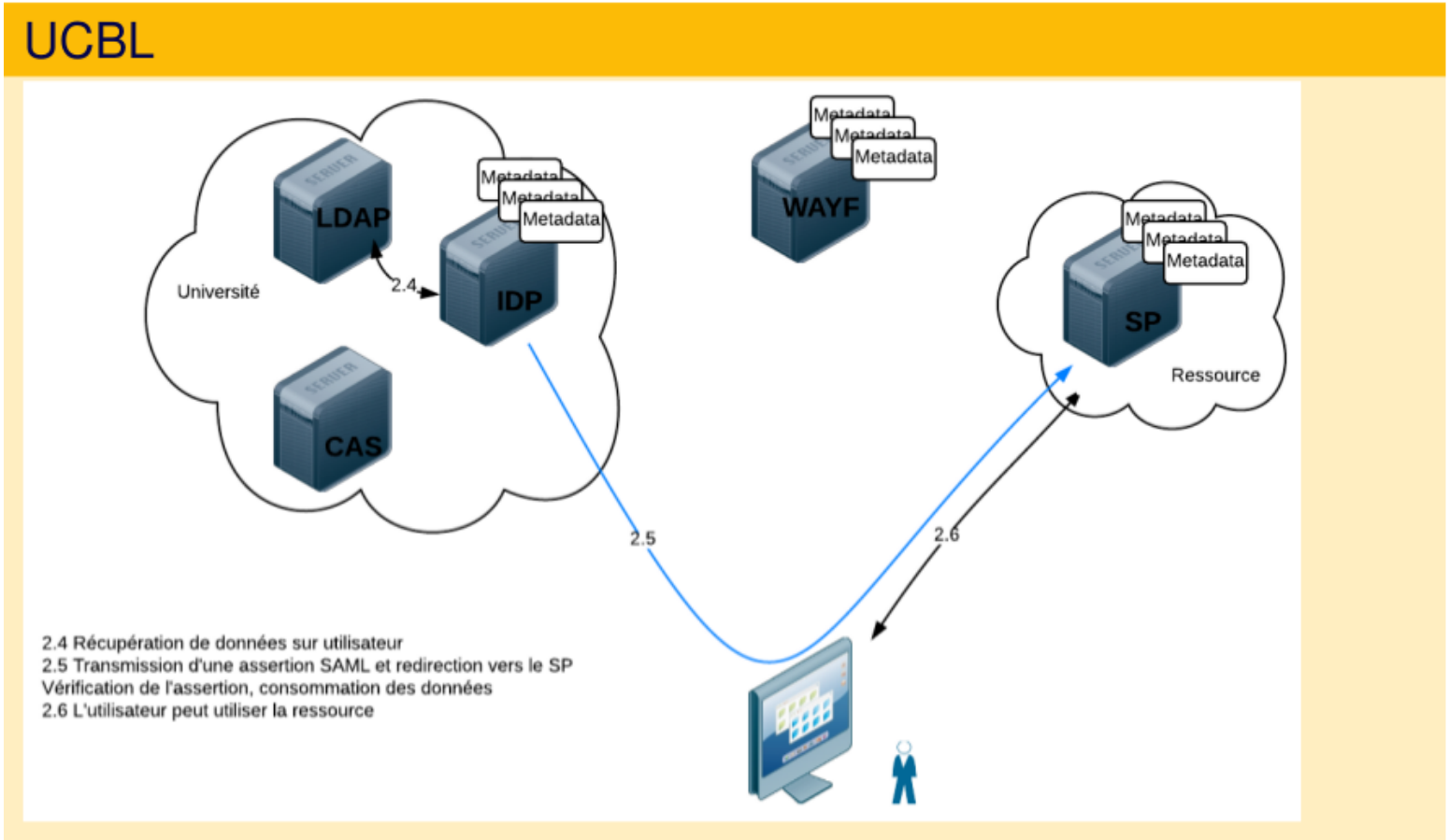
Les métadonnées sont gérées et transmise par la fédération ce qui permet à 2 entités de la fédération de maintenir la confiance entre elles.

Where are you from

Dans une fédération, c'est l'entité d'origine qui authentifie un utilisateur.
Pour accéder à un service, ce dernier doit donc dire quel entité contacter.
C'est le role du Where are you from (WAYF).



Récupération des informations



? Comment les informations sont-elle vérifiées ?

? Comment les informations peuvent-elle être chiffrées ?

Pourquoi allez plus loin

Le système de fédération suppose :

- un accord entre des entité ;
- une confiance entre ces entité ;
- une structure pour gérer les métadonnées.

Sur le web, ce n'est pas toujours possible de creer une fédération contenant les IDP (Facebook, google, ...) et tous les SP (les jeux, picassa, ...).
Il faut donc un moyen plus souple où l'utilisateur peut décider s'il partage des informations et celles qu'il partage.

OpenID

Il n'est plus question de fédération, il n'y pas d'échange de métadonnées. L'utilisateur lors de l'authentification décide d'autoriser un SP à obtenir des informations personnelles auprès de l'IDP.

Quelques dates

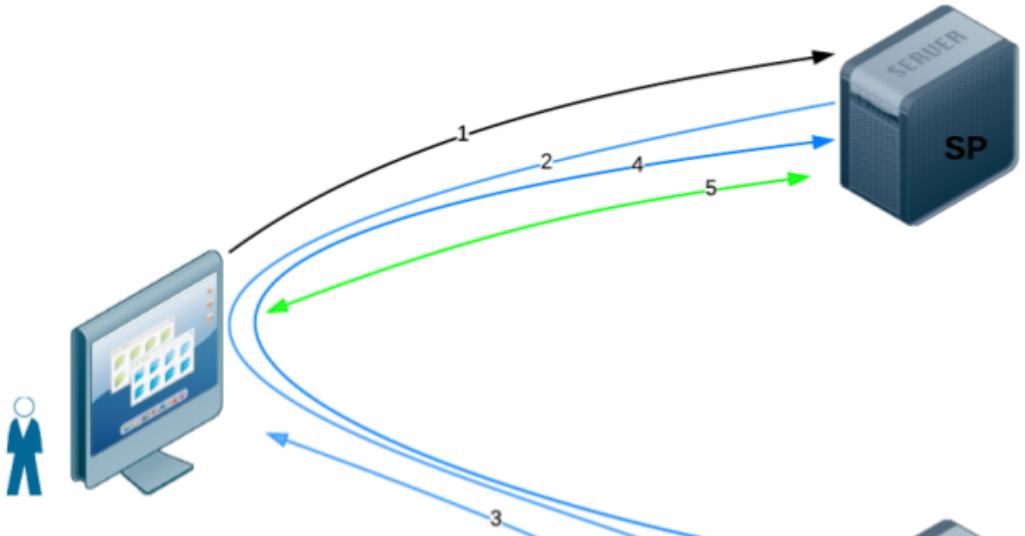
2005 Brad Fitzpatrick (LiveJournal - SixApart)

2007 Symantec, Microsoft - Compatibilité CardSpace

2007 OpenID Foundation (Google, IBM, Microsoft, Verisign, etc)

2008 IDP publics (Yahoo !, Soureforge, Google, etc

OpenID : Fonctionnement



1. L'utilisateur demande à accéder à une ressource utilisant open ID, il saisie son openID
   Un openID est une URL : **openid.univ-lyon1.fr/p123456**
2. Le SP le redirige vers l'IDP openID (URL) et intègre sa demande d'information
3. L'IDP authentifie l'utilisateur et demande s'il autorise le SP à accèder aux informations demandées
4. L'IDP intègre les informations demandées et redirige l'utilisateur vers le SP

Le SP et l'IDP communique de façon chiffré par clef partagée (Diffie–Hellman)

OptenID : avenir

- OpenID est une technologie récente mais qui a tendance à disparaitre
- Les ≪ gros ≫ fournisseurs d'indentités (google, facebook, ...) ne propose plus ce service.
- Remplacement par OpenID-Connect qui, malgrès sont nom, n'a pas vraiment de lien.

La durée de vie de ces technologies est l'un de leur plus gros défaut.

Utilisation d'un API

On peut allez plus loin en créant un token authorisant une application à agir en votre nom sur une API spécifique de l'IDP
Par exemple

- Un programme de votre téléphone qui gère vos mails, vos rendez-vous...
- Un appareil photo qui ajoute automatiquement les photos sur votre compte.
- Des robots qui recherchent automatiquement des tweets selon leur utilisateur ou des mots clefs.

C'est ce que permet le protocole OAuth.

Pourquoi ne pas simplement stocket vos login/mot de passe dans l'application cliente ?

OAuth

Définition (Pour les applications - API)

Le protocole OAuth permet à un utilisateur d'autoriser une application tierce à accéder à ses propres ressources détenues par un service. Cela sans confier son login/mot de passe à l'application.

Quelques dates

2006 Debuts des travaux

2007 OAuth Core 1.0

2010 OAuth Core 1.0a = RFC 5849

2012 OAuth 2.0 = RFC 6749 & RFC 6750

OAuth

Vocabulaire

- *Resource owner* : utilisateur concerné.
- *Resource server ou provider* : service qui stocke les informations utilisateur a propose l'API.
- *Resource consumer ou Client* : application tierce qui souhaite accéder à l'API.
- *Authorization server* : serveur qui délivre les jetons. Souvent c'est aussi le *Resource server*

- OAuth 2.0 n'est pas rétro compatible OAuth 1.0
- Problème de gouvernance, mais standard de fait

Enregistrement du client

Un *Ressource consumer* qui souhaite utiliser l'API d'un *Ressource provider* a besoin d'être enregistré :

- L'enregistrement demande :
  - un nom ;
  - des urls qui seront utilisées par le serveur d'autorisation pour fournir les tokens ;
  - les types d'autorisations qui seront demandées ;
  - le nom de domaine qui pourra effectuer les requète vers le serveur de ressource.

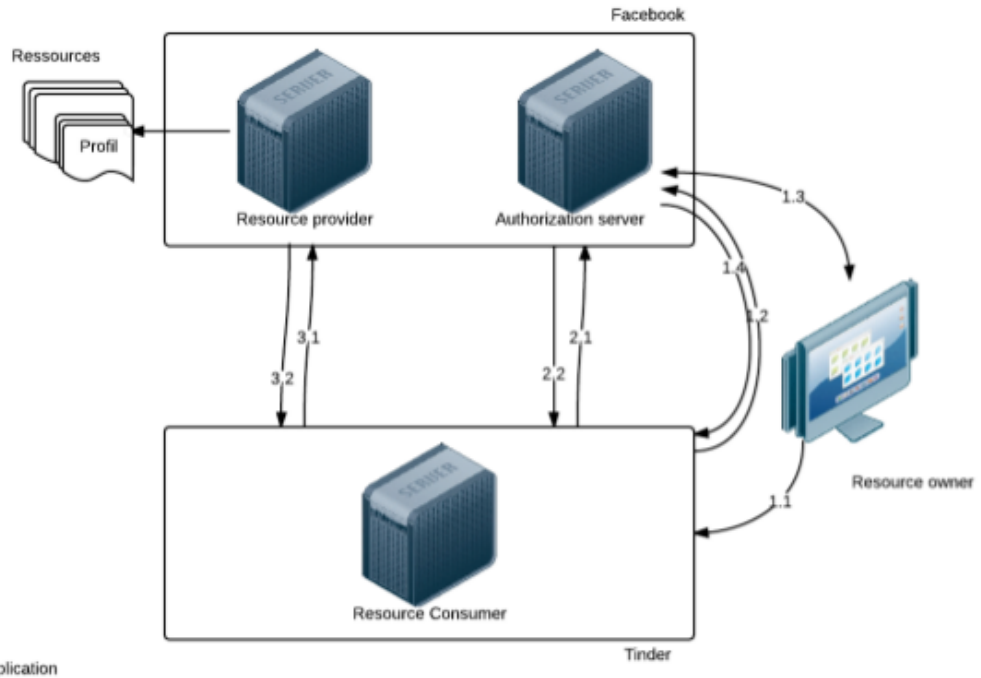
A cette occasion, les 2 entités échangent une clef secrète qui sera utilisée pour chiffrer ou signer les échanges.

Authorisation

Il y a plusieurs types d'autorisations :

- Autorisation via un code : le propriétaire autorise l'accès et le token est transmis directement par le serveur d'autorisation au client. LE token est renouvelable.
- Autorisation implicite : moins sécurisée, le token est transmis au client.
- Autorisation via le mot de passe : comme avant, les *credentials* sont transmises au *ressource consumer*.
- Autorisation entre serveur : si le client est aussi détenteur des données, l'utilisateur final n'est pas informé.

OpenID exemples



1.1 : Accès à l'application
1.2 : Redirection HTTP vers l'AS
1.3 : Authentification, délégation, autorisation
1.4 : Redirection vers l'application + Code
2.1 : Authentification du RC (client (API Key) + Code)
2.2: Jeton d'accès (access_token)
3.1 : Requête de récupération du profil
3.2 : Récupération du profil

OpenID Connect OIDC

Authentification et autorisations

Evolutions permettant l'authentification et l'autorisation pour des services Web, applications mobiles ou web clientes.

- OpenID Connect, couche d'authentification au dessus d'OAuth 2.0, standard depuis Février 2014
- Soutenu par l'OpenID Foundation
- Implémentations : Google, Microsoft, Ping Identity, MitreID-Connect, Gluu, OpenAM

OpenID Connect OIDC

Techniquement

- HTTPS + REST API + JSON
- Protocoles extensibles : OpenID Provider disovery, dynamic registration, session management, etc
- Successeur du SAML ?

Conclusion

- Authentification des serveur par certificat x509
  - des défauts ;
  - mal maitrisé par le grand publique ;
  - mais à la base de beaucoup de modèles de sécurité.
- Grosse évolution pour l'authentification déporté
  - des standard de fait ;
  - pas de rétro-compatibilité (pourquoi ?) ;
  - possibilité de déléguer les actions.

Bibliographie


Touradj Ebrahimi, Franck Leprévost, and Bertrand Warusfel.
Cryptographie et sécurité des systèmes et réseaux.
hermes, 2006.


Thierry Harlé and Florent Skrabacz.
Clés pour la sécurité des SI.
hermes, 2004.


Johann Reinke.
Comprendre oauth2, January 2016.
http ://www.bubblecode.net/fr/2016/01/22/comprendre-oauth2/.


William Stallings.
Sécurité des réseaux, Applications et Standards.
vuibert, 2002.

Exemple de questions

- D'après le cours, pour l'authentification web non centralisées (lorsque plusieurs groupes partagent les données des utilisteurs), il y a 2 familles de protocoles :
  - fédérative : shibboleth, SAML ...
  - centrée utilisateur : openid, oauth ...

  Vous êtes une entreprise programmant des jeu sur tablette et souhaitez obtenir les informations du compte faceboook des joueurs. Quelle famille de protocole utilisez-vous ? Pourquoi ? Que permettent de faire ces protocoles ?
- Pour permettre à mon télépone de lire les mails arrivés sur le serveur yooha.fr, je dispose de 2 méthodes :
  - le login/mot de passe envoyé via un canal crypté (SSL/TLS) ;
  - une authentification via oauth2.

  Sachant que mon téléphone pourra dans le futur être compromis ou volé, quel méthode trouvez-vous la plus sécurisée ? Justifiez bien sur la réponse.
- Quels services sont nécessaires à la mise en place d'un serveur CAS. Parmis eux, lesquels doivent absolument utiliser du chiffrement et pourquoi (pour l'authentification, le confidientialité et/ou la non falsification). À chaque fois justifier la raison de ce besoin en mentionnant une attaque possible.