SÉCURITÉ DES RÉSEAUX
AUTHENTIFICATION DES UTILISATEURS

A. Guermouche

Plan

Plan

Single Sign On

SSO

Avec le SSO, un utilisateur n'a qu'à saisir ses identifiants de connexion (nom d'utilisateur, mot de passe, etc.) une seule fois pour accéder à toutes ses applications.

Intérêt

- Des mots de passe plus forts : Comme les utilisateurs n'ont à utiliser qu'un seul mot de passe, le SSO leur permet de créer, de mémoriser et d'utiliser plus facilement des mots de passe plus forts.
- Pas de mots de passe répétés : Vu qu'un seul mot de passe va donner accès à plusieurs service, l'utilisateur ne va pas utiliser le même mot de passe pour différents services.
- Point unique pour la gestion des mots de passe : Avec le SSO, la politique de sécurité et de gestion du mot est plus facile à maintenir vu qu'elle ne concerne qu'un seul identifiant.

Fonctionnement

Processus d'authentification

- Chaque fois qu'un utilisateur se connecte à un service de SSO, le service crée un jeton d'authentification qui rappelle que l'utilisateur est vérifié.
- Un jeton d'authentification est une une carte d'identité temporaire délivrée à l'utilisateur.
- L'utilisateur doit fournir le jeton à chaque fois qu'il veut accéder à un service. Le serveur concerné pourra alors vérifier le jeton en interagissant avec le service de SSO.

2 services de SSO seront considérés:

- Kerberos pour un mécanisme de SSO dans un réseau local.
- WebSSO pour un mécanisme de SSO pour applications web.

Plan

Présentation

Kerberos est un service d'authentification développé par comme partie du projet Athena du MIT.

Problématique :

- Comment assurer l'authentification pour les demandes de services dans un environnement distribué ouvert?
- Problème de confiance en les mécanismes d'authentification de chacun des postes de travail.
  - Un utilisateur peut obtenir l'accès à un poste de travail et feindre d'être un autre utilisateur.
  - Un utilisateur peut changer l'adresse réseau d'un poste de travail pour que les demandes envoyées depuis ce dernier semblent venir d'un autre poste de travail.
  - Un utilisateur peut espionner des échanges et lancer une attaque par rejeu pour perturber des opérations.
- Kerberos fournit un serveur centralisé d'authentification dont la fonction est d'authentifier les utilisateurs vis-à-vis des serveurs et inversement.

Exigences

Le premier rapport publié sur Kerberos énumère les exigences suivantes :

sûr. un espion sur le réseau ne doit pas pouvoir obtenir l'information nécessaire pour se faire passer pour un utilisateur.

fiable. Kerberos doit pouvoir reposer sur une architecture de serveurs distribuée avec des systèmes pouvant en remplacer d'autres.

transparent. idéalement, l'utilisateur ne doit pas être conscient du processus d'authentification.

évolutif. le système doit être capable de gérer un grand nombre de clients et de serveurs.

Le schéma complet de Kerberos est celui d'un service d'authentification par un tiers de confiance utilisant un protocole s'inspirant de celui de Needham Schroeder.

Rappel sur le protocole de Needham Schroeder

Soient A et B deux systèmes communiquant via un réseau pas sûr devant déterminer une clé de session.

Notations :

S. est un serveur de confiance pour A et B.

K_{AS}. clé symétrique connue uniquement par A et S.

K_{BS}. clé symétrique connue uniquement par B et S.

N_A et N_B. des nonces.

Protocole :

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

Protocole d'authentification de Kerberos (1/5)

- Utilisation d'un serveur d'authentification (SA) qui connaît tous les mots de passe.
- Le SA partage une clé secrète unique avec chaque serveur.

Lorsqu'un utilisateur demande un service à un serveur :

1. Le module client du poste de travail de l'utilisateur envoie le mot de passe saisi au SA dans un message contenant l'ID de l'utilisateur, son mot de passe et l'ID du serveur.

Protocole d'authentification de Kerberos (1/5)

- Utilisation d'un serveur d'authentification (SA) qui connaît tous les mots de passe.
- Le SA partage une clé secrète unique avec chaque serveur.

Lorsqu'un utilisateur demande un service à un serveur :

1. Le module client du poste de travail de l'utilisateur envoie le mot de passe saisi au SA dans un message contenant l'ID de l'utilisateur, son mot de passe et l'ID du serveur.
2. Le SA authentifie l'utilisateur.

Protocole d'authentification de Kerberos (1/5)

- Utilisation d'un serveur d'authentification (SA) qui connaît tous les mots de passe.
- Le SA partage une clé secrète unique avec chaque serveur.

Lorsqu'un utilisateur demande un service à un serveur :

1. Le module client du poste de travail de l'utilisateur envoie le mot de passe saisi au SA dans un message contenant l'ID de l'utilisateur, son mot de passe et l'ID du serveur.
2. Le SA authentifie l'utilisateur.
3. Le SA génère un ticket contenant l'ID de l'utilisateur et l'ID du serveur et le chiffre avec la clé secrète partagée par le SA et le serveur.

Protocole d'authentification de Kerberos (1/5)

- Utilisation d'un serveur d'authentification (SA) qui connaît tous les mots de passe.
- Le SA partage une clé secrète unique avec chaque serveur.

Lorsqu'un utilisateur demande un service à un serveur :

1. Le module client du poste de travail de l'utilisateur envoie le mot de passe saisi au SA dans un message contenant l'ID de l'utilisateur, son mot de passe et l'ID du serveur.
2. Le SA authentifie l'utilisateur.
3. Le SA génère un ticket contenant l'ID de l'utilisateur et l'ID du serveur et le chiffre avec la clé secrète partagée par le SA et le serveur.
4. Le ticket est envoyé au poste client.

Protocole d'authentification de Kerberos (1/5)

- Utilisation d'un serveur d'authentification (SA) qui connaît tous les mots de passe.
- Le SA partage une clé secrète unique avec chaque serveur.

Lorsqu'un utilisateur demande un service à un serveur :

1. Le module client du poste de travail de l'utilisateur envoie le mot de passe saisi au SA dans un message contenant l'ID de l'utilisateur, son mot de passe et l'ID du serveur.
2. Le SA authentifie l'utilisateur.
3. Le SA génère un ticket contenant l'ID de l'utilisateur et l'ID du serveur et le chiffre avec la clé secrète partagée par le SA et le serveur.
4. Le ticket est envoyé au poste client.

Remarque :

- Le ticket contient l'ID du serveur pour qu'il puisse vérifier qu'il a correctement déchiffrer le ticket,
- l'ID du client pour dire que le ticket a été émis par le module client C
- et l'adresse réseau du poste client (ou le nom) pour éviter l'usurpation d'identité en passant par une autre machine.

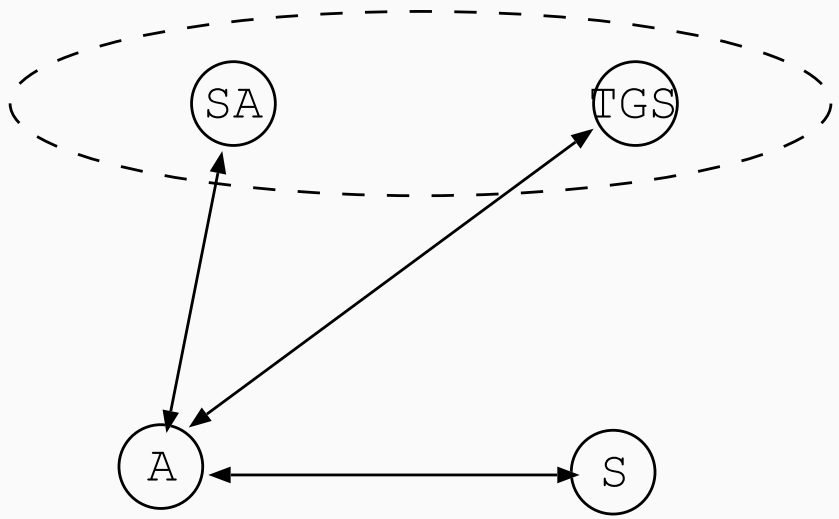
Protocole d'authentification de Kerberos (2/5)

- Réduire le nombre de fois où l'utilisateur doit saisir son mot de passe.
- Le mot de passe du client transitait en clair sur le réseau.

Protocole d'authentification de Kerberos (2/5)

- Réduire le nombre de fois où l'utilisateur doit saisir son mot de passe.
- Le mot de passe du client transitait en clair sur le réseau.

Solution : Utilisation d'un serveur d'octroi de ticket (TGS).



Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$

Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$
2. $SA \rightarrow A : \{Ticket_{TGS}\}_{mdp}$

Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$
2. $SA \rightarrow A : \{Ticket_{TGS}\}_{mdp}$
3. $A \rightarrow TGS : A, S, Ticket_{TGS}$

Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$
2. $SA \rightarrow A : \{Ticket_{TGS}\}_{mdp}$
3. $A \rightarrow TGS : A, S, Ticket_{TGS}$
4. $TGS \rightarrow A : Ticket_S$

Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$
2. $SA \rightarrow A : \{Ticket_{TGS}\}_{mdp}$
3. $A \rightarrow TGS : A, S, Ticket_{TGS}$
4. $TGS \rightarrow A : Ticket_S$
5. $A \rightarrow S : A, Ticket_S$

Protocole d'authentification de Kerberos (3/5)

Protocole

1. $A \rightarrow SA : A, TGS$
2. $SA \rightarrow A : \{Ticket_{TGS}\}_{mdp}$
3. $A \rightarrow TGS : A, S, Ticket_{TGS}$
4. $TGS \rightarrow A : Ticket_S$
5. $A \rightarrow S : A, Ticket_S$

où:
$Ticket_{TGS}$ ou TGT = $\{A, addr_A, TGS, horrodatage\}_{K_{SA,TGS}}$
et
$Ticket_S = \{A, S, addr_C, horrodatage\}_{K_{TGS,S}}$

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.
2. Le SA répond par un ticket chiffré avec une clé dérivée du mot de passe de l'utilisateur. Quand cette réponse arrive, le client demande son mot de passe à l'utilisateur, génère la clé et essaye de déchiffrer le ticket (seul l'utilisateur légitime est capable de déchiffrer ce message).

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.
2. Le SA répond par un ticket chiffré avec une clé dérivée du mot de passe de l'utilisateur. Quand cette réponse arrive, le client demande son mot de passe à l'utilisateur, génère la clé et essaye de déchiffrer le ticket (seul l'utilisateur légitime est capable de déchiffrer ce message).
3. Le client demande un ticket d'octroi de service de la part de l'utilisateur au TGS (le message correspondant contient l'ID de l'utilisateur , l'ID du service, et le ticket d'octroi de tickets).

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.
2. Le SA répond par un ticket chiffré avec une clé dérivée du mot de passe de l'utilisateur. Quand cette réponse arrive, le client demande son mot de passe à l'utilisateur, génère la clé et essaye de déchiffrer le ticket (seul l'utilisateur légitime est capable de déchiffrer ce message).
3. Le client demande un ticket d'octroi de service de la part de l'utilisateur au TGS (le message correspondant contient l'ID de l'utilisateur , l'ID du service, et le ticket d'octroi de tickets).
4. Le TGS déchiffre le ticket et vérifie le succès du déchiffrement par la présence de son ID. Si tout est bon, il génère un ticket pour accorder l'accès au service.

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.
2. Le SA répond par un ticket chiffré avec une clé dérivée du mot de passe de l'utilisateur. Quand cette réponse arrive, le client demande son mot de passe à l'utilisateur, génère la clé et essaye de déchiffrer le ticket (seul l'utilisateur légitime est capable de déchiffrer ce message).
3. Le client demande un ticket d'octroi de service de la part de l'utilisateur au TGS (le message correspondant contient l'ID de l'utilisateur , l'ID du service, et le ticket d'octroi de tickets).
4. Le TGS déchiffre le ticket et vérifie le succès du déchiffrement par la présence de son ID. Si tout est bon, il génère un ticket pour accorder l'accès au service.
5. Le client demande l'accès à un service de la part de l'utilisateur (transmission d'un message contenant l'ID de l'utilisateur et le ticket d'octroi de service).

Protocole d'authentification de Kerberos (4/5)

1. Le client demande un ticket d'octroi de tickets de la part de l'utilisateur au SA.
2. Le SA répond par un ticket chiffré avec une clé dérivée du mot de passe de l'utilisateur. Quand cette réponse arrive, le client demande son mot de passe à l'utilisateur, génère la clé et essaye de déchiffrer le ticket (seul l'utilisateur légitime est capable de déchiffrer ce message).
3. Le client demande un ticket d'octroi de service de la part de l'utilisateur au TGS (le message correspondant contient l'ID de l'utilisateur , l'ID du service, et le ticket d'octroi de tickets).
4. Le TGS déchiffre le ticket et vérifie le succès du déchiffrement par la présence de son ID. Si tout est bon, il génère un ticket pour accorder l'accès au service.
5. Le client demande l'accès à un service de la part de l'utilisateur (transmission d'un message contenant l'ID de l'utilisateur et le ticket d'octroi de service).

Remarque :

- Inclusion d'un horodatage dans le ticket d'octroi de tickets pour éviter le rejeu.
- Le ticket de service est chiffré avec une clé connue uniquement du TGS et du serveur (Le ticket d'octroi est lui chiffré par une clé connue par le SA et le TGS).

Protocole d'authentification de Kerberos (5/5)

Problème du protocole précédent

- Problème de durée de vie du ticket d'octroi de ticket et octroi de service (ainsi que l'authentification de la personne utilisant le ticket vis à vis du ticket).
- Authentification des serveurs vis à vis des clients.

Solution :

- Le SA fournit à la fois au TGS et au client une information secrète de façon sûre (utilisation d'une clé de session).

Protocole d'authentification de Kerberos (5/5)

Problème du protocole précédent

- Problème de durée de vie du ticket d'octroi de ticket et octroi de service (ainsi que l'authentification de la personne utilisant le ticket vis à vis du ticket).
- Authentification des serveurs vis à vis des clients.

Solution :

- Le SA fournit à la fois au TGS et au client une information secrète de façon sûre (utilisation d'une clé de session).
  - La distribution de la clé se fait par le biais d'un message, du SA vers le client, contenant la clé de session ainsi que le ticket et chiffré avec une clé dérivée du mot de passe.

Protocole d'authentification de Kerberos (5/5)

Problème du protocole précédent

- Problème de durée de vie du ticket d'octroi de ticket et octroi de service (ainsi que l'authentification de la personne utilisant le ticket vis à vis du ticket).
- Authentification des serveurs vis à vis des clients.

Solution :

- Le SA fournit à la fois au TGS et au client une information secrète de façon sûre (utilisation d'une clé de session).
  - La distribution de la clé se fait par le biais d'un message, du SA vers le client, contenant la clé de session ainsi que le ticket et chiffré avec une clé dérivée du mot de passe.
  - Cette même clé est incluse dans le ticket (qui ne peut être lu que par le TGS).

Protocole d'authentification de Kerberos (5/5)

Problème du protocole précédent

- Problème de durée de vie du ticket d'octroi de ticket et octroi de service (ainsi que l'authentification de la personne utilisant le ticket vis à vis du ticket).
- Authentification des serveurs vis à vis des clients.

Solution :

- Le SA fournit à la fois au TGS et au client une information secrète de façon sûre (utilisation d'une clé de session).
  - La distribution de la clé se fait par le biais d'un message, du SA vers le client, contenant la clé de session ainsi que le ticket et chiffré avec une clé dérivée du mot de passe.
  - Cette même clé est incluse dans le ticket (qui ne peut être lu que par le TGS).
  - La clé est utilisée par la suite pour vérifier l'identité du client via un "authentificateur".

Protocole d'authentification de Kerberos (5/5)

Problème du protocole précédent

- Problème de durée de vie du ticket d'octroi de ticket et octroi de service (ainsi que l'authentification de la personne utilisant le ticket vis à vis du ticket).
- Authentification des serveurs vis à vis des clients.

Solution :

- Le SA fournit à la fois au TGS et au client une information secrète de façon sûre (utilisation d'une clé de session).
  - La distribution de la clé se fait par le biais d'un message, du SA vers le client, contenant la clé de session ainsi que le ticket et chiffré avec une clé dérivée du mot de passe.
  - Cette même clé est incluse dans le ticket (qui ne peut être lu que par le TGS).
  - La clé est utilisée par la suite pour vérifier l'identité du client via un "authentificateur".
- Cette même technique est utilisée entre client et le serveur (le TGS générant une clé de session).

Royaumes Kerberos

Un “royaume” Kerberos est l’environnement contenant un serveur et un ensembles de clients. Le royaume nécessite :

- Le serveur Kerberos doit connaître l’ID utilisateur (UID) et le mot de passe haché de tous les utilisateurs.
- Le serveur Kerberos doit partager une clé secrète avec chaque serveur. Tous les serveurs sont enregistrés sur le serveur Kerberos.

Le royaume est une notion centrale dans la configuration des différentes versions de serveurs Kerberos (MIT, Heimdal, . . .).

Plan

WebSSO

Principe de base

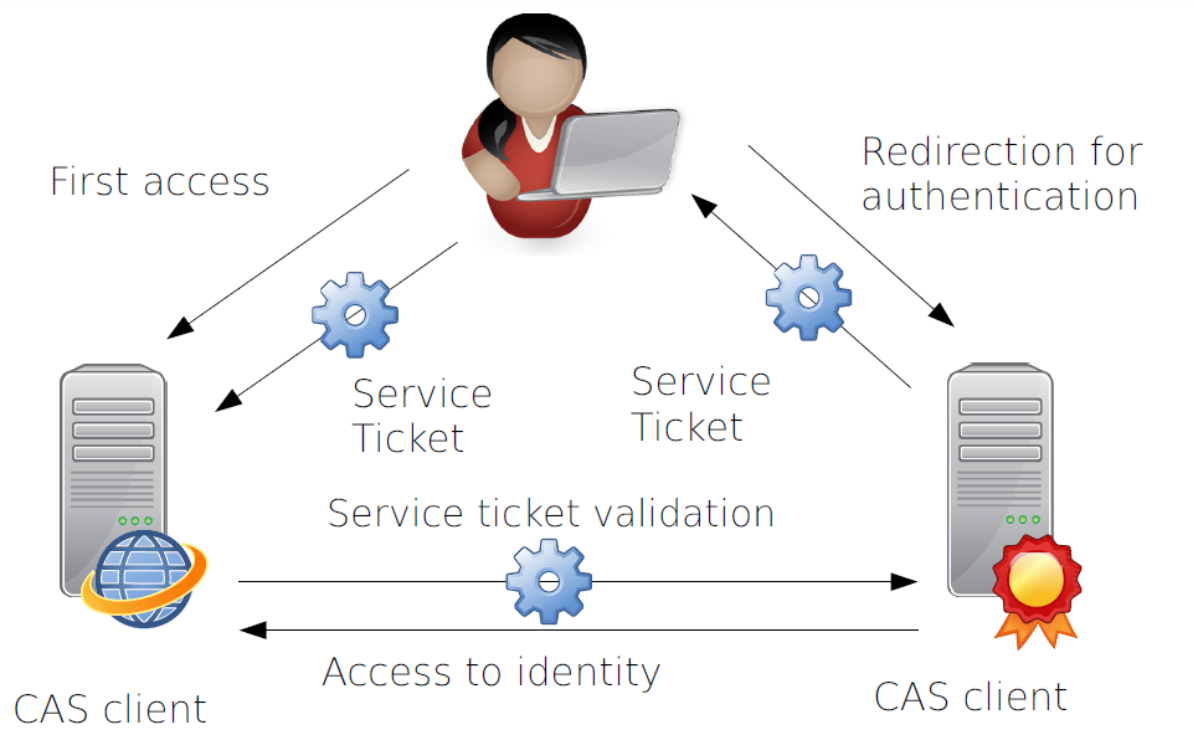
Intercepter les requêtes entre le client et le serveur, et indiquer au serveur que le client est bien authentifié.

Fédération d'identités

- Notions de cercle de confiance, fournisseur d'identités (IDP) et fournisseur de service (SP)
- L'utilisateur qui possède plusieurs identités numériques peut les fédérer au sein d'un cercle de confiance
- Le résultat visible est l'accès transparent aux fournisseurs de service, mais d'autres avantages existent, comme la déconnexion unique (SLO)

CAS

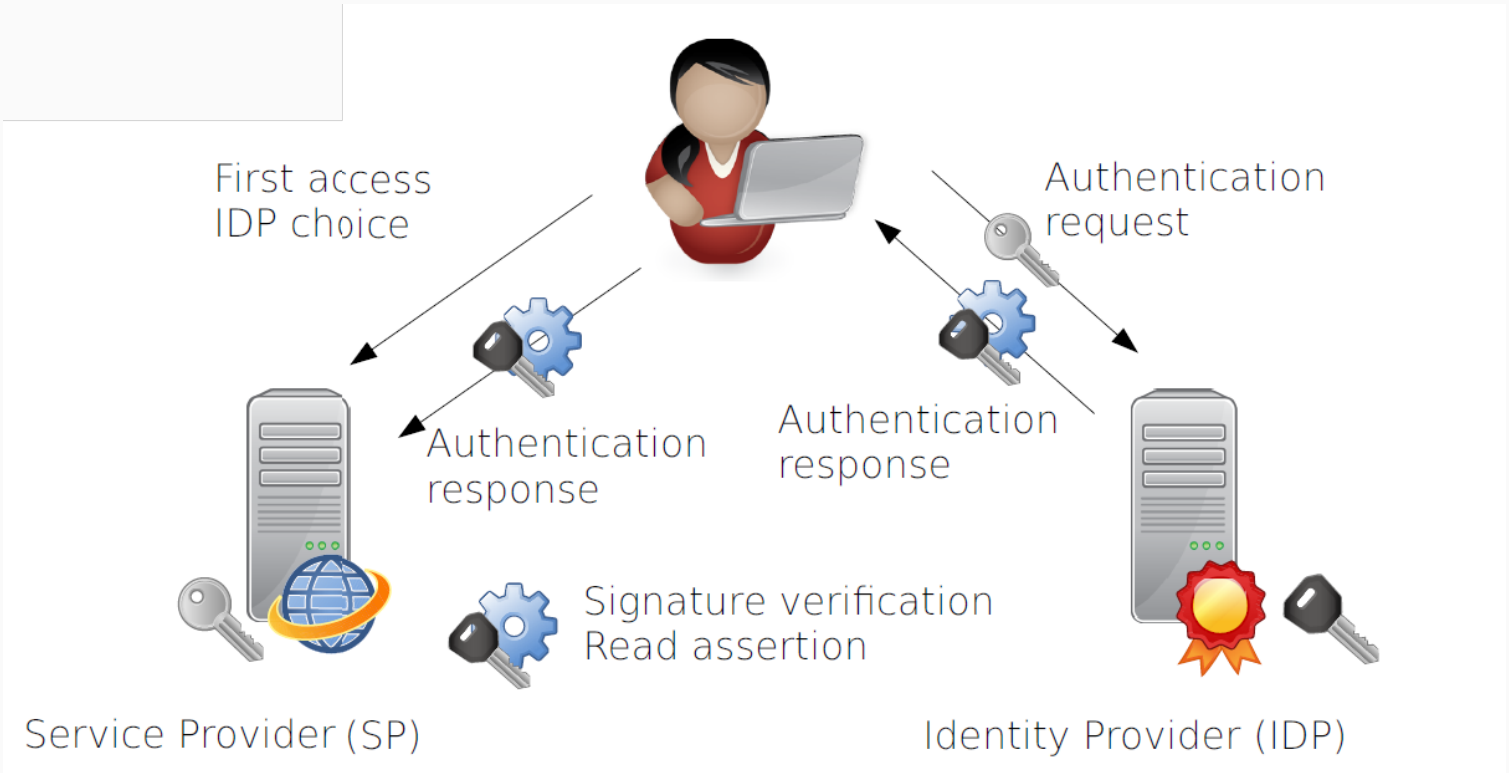
- Cible les réseaux d'entreprise.
- Pas d'obligation pour le serveur CAS de connaître les clients (la confiance est basée sur le certificat du serveur).



source https://ldapcon.org/2017/wp-content/uploads/2017/08/16_Cl%C3%A9ment-Oudot_PRE_LDAPCon2017_SSO-1.pdf

SAML

- Vise principalement les réseaux d'entreprise.
- Les fournisseurs d'identité (IDP) et les fournisseurs de services (SP) doivent être enregistrés.

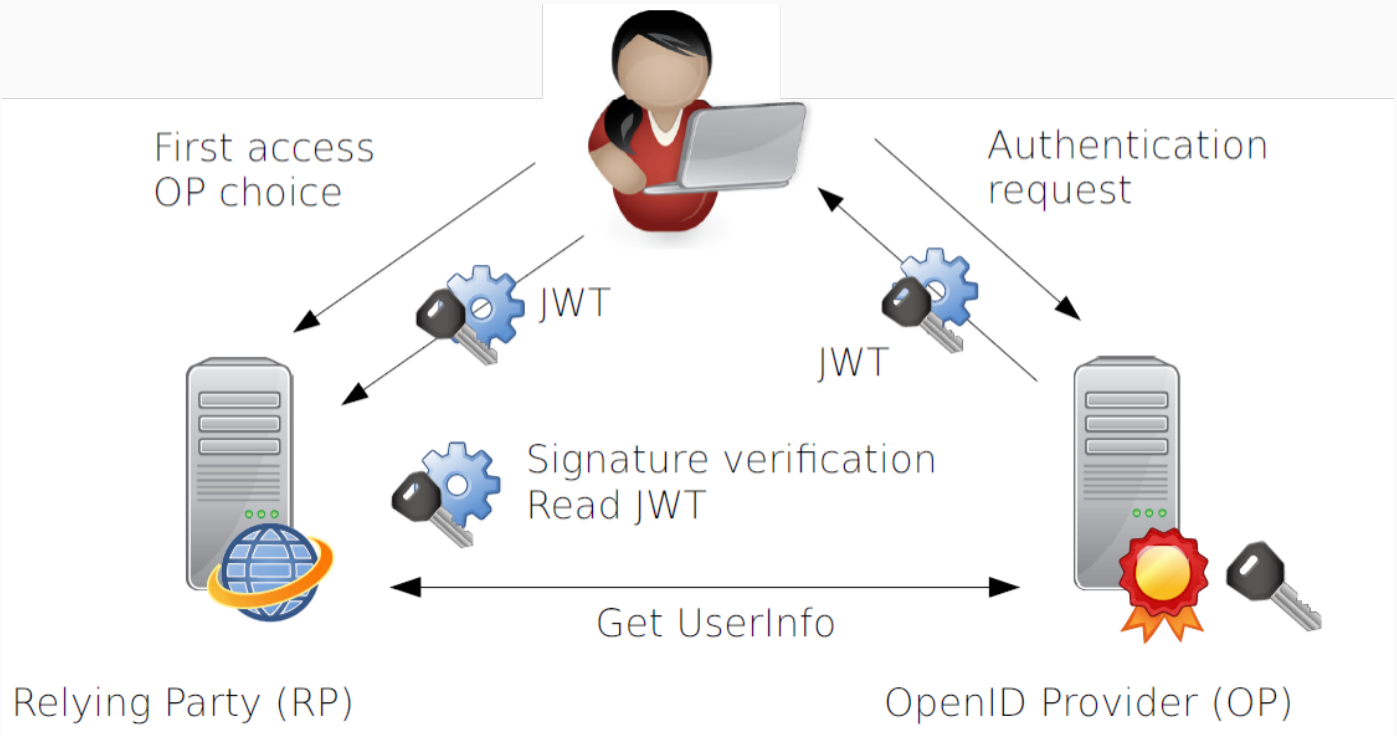


source https://ldapcon.org/2017/wp-content/uploads/2017/08/16_C1%C3%A9ment-Oudot_PRE_LDAPCon2017_SSO-1.pdf

OpendID Connect

OpenID

- Est largement utilisé pour permettre aux utilisateurs de se connecter à des sites web grand public et à des applications mobiles.
- Est basé sur OAuth2 (standard pour la délégation d'identité et JWT (*JSON Web Token*)



source https://ldapcon.org/2017/wp-content/uploads/2017/08/16_Cl%C3%A9ment-Oudot_PRE_LDAPCon2017_SSO-1.pdf

Plan

Oauth2

Objectif

Comment une application peut-elle accéder à une ressource protégée au nom de son propriétaire sans connaître ses identifiants (login et mot de passe)?

Exemple. Je veux autoriser mon application préférée à utiliser mon compte Google.

Oauth2 est un mécanisme de délégation d'autorisation.

Rappel

L'authentification est liée à l'identité alors que l'autorisation est liée aux privilèges.

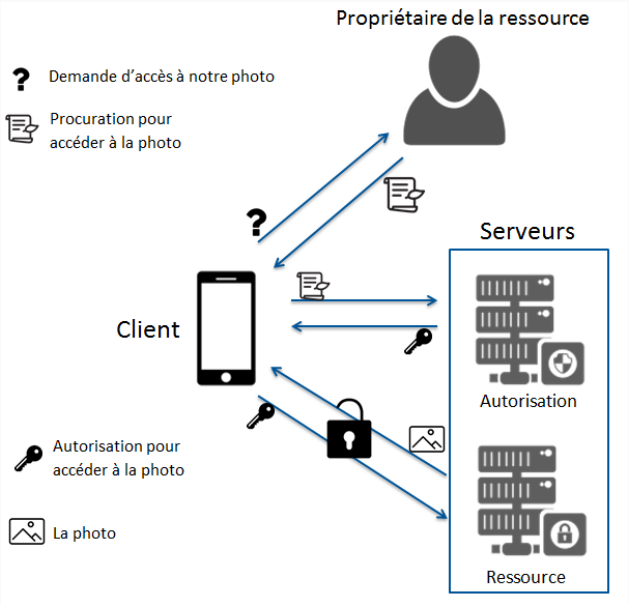
Fonctionnement

Flots d'autorisation

OAuth définit 4 types de flots d'autorisation (qui sont plus ou moins sécurisés) :

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

Oauth2 fait intervenir 4 entités: client, propriétaire (l'utilisateur), le serveur et le serveur d'autorisation.



source https://zestedesavoir.com/articles/1616/comprendre-oauth-2-0-par-lexemple/

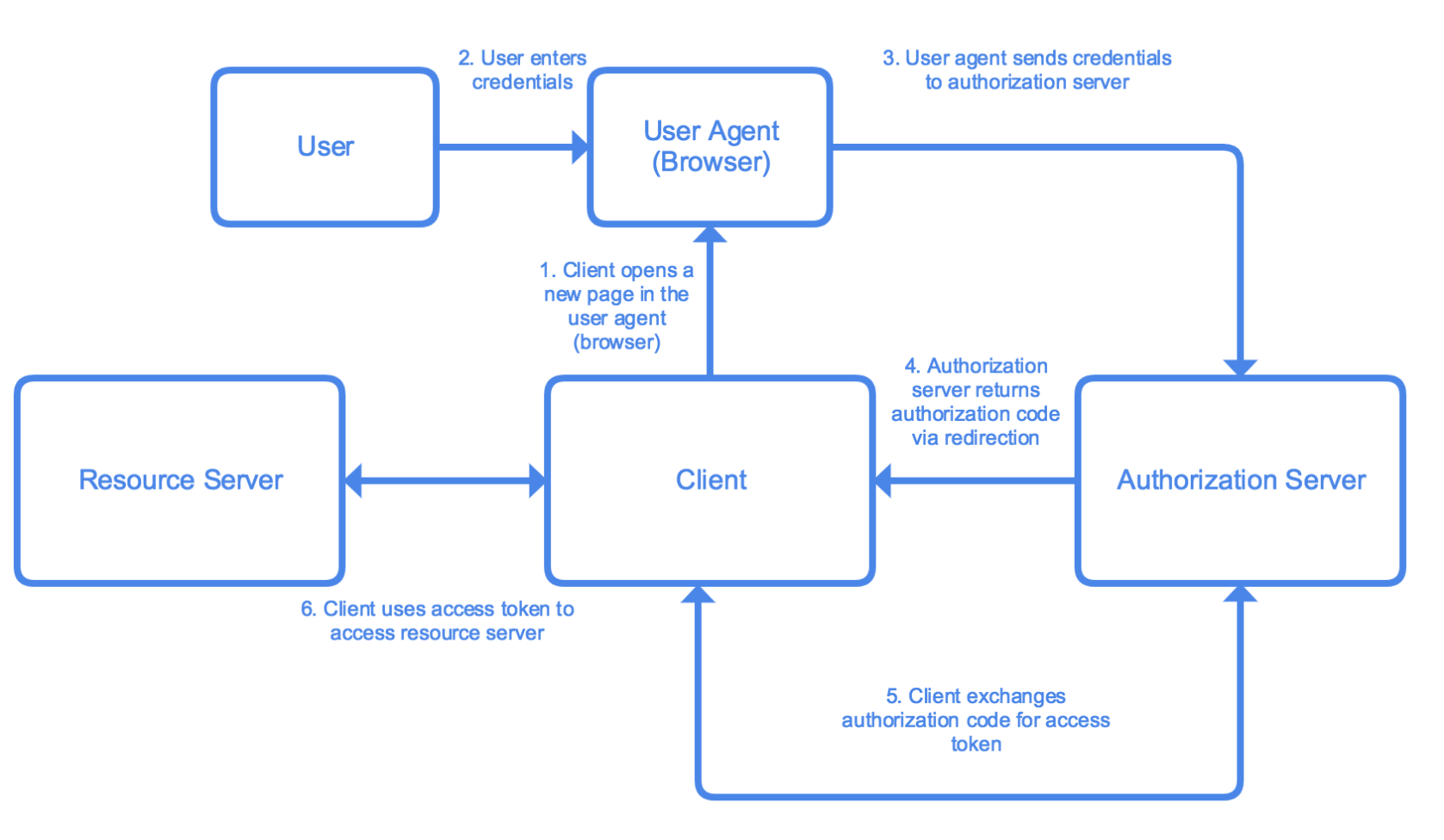
Authorization Code (1/2)

Fonctionnement

1. Le client redirige le propriétaire de la ressource vers le serveur d'autorisation. Le client doit inclure son identifiant dans la requête de redirection et le niveau d'accès qu'il souhaite obtenir.
2. Le propriétaire de la ressource s'authentifie auprès du serveur d'autorisation et approuve ou non la requête du client.
3. Le serveur d'autorisation redirige le propriétaire de la ressource en utilisant l'URL de redirection défini par le client.
4. Avec le code d'autorisation ainsi obtenu, le client demande un token d'accès en prenant le soin de s'authentifier à son tour auprès du serveur d'autorisation.
5. Le client utilise le token d'accès pour accéder au serveur.

Dans ce scénario le client doit être capable de conserver le code d'autorisation de manière sûre.

Authorization Code (2/2)



source https://cloud.google.com/community/tutorials/understanding-oauth2-and-deploy-a-basic-auth-srv-to-cloud-functions

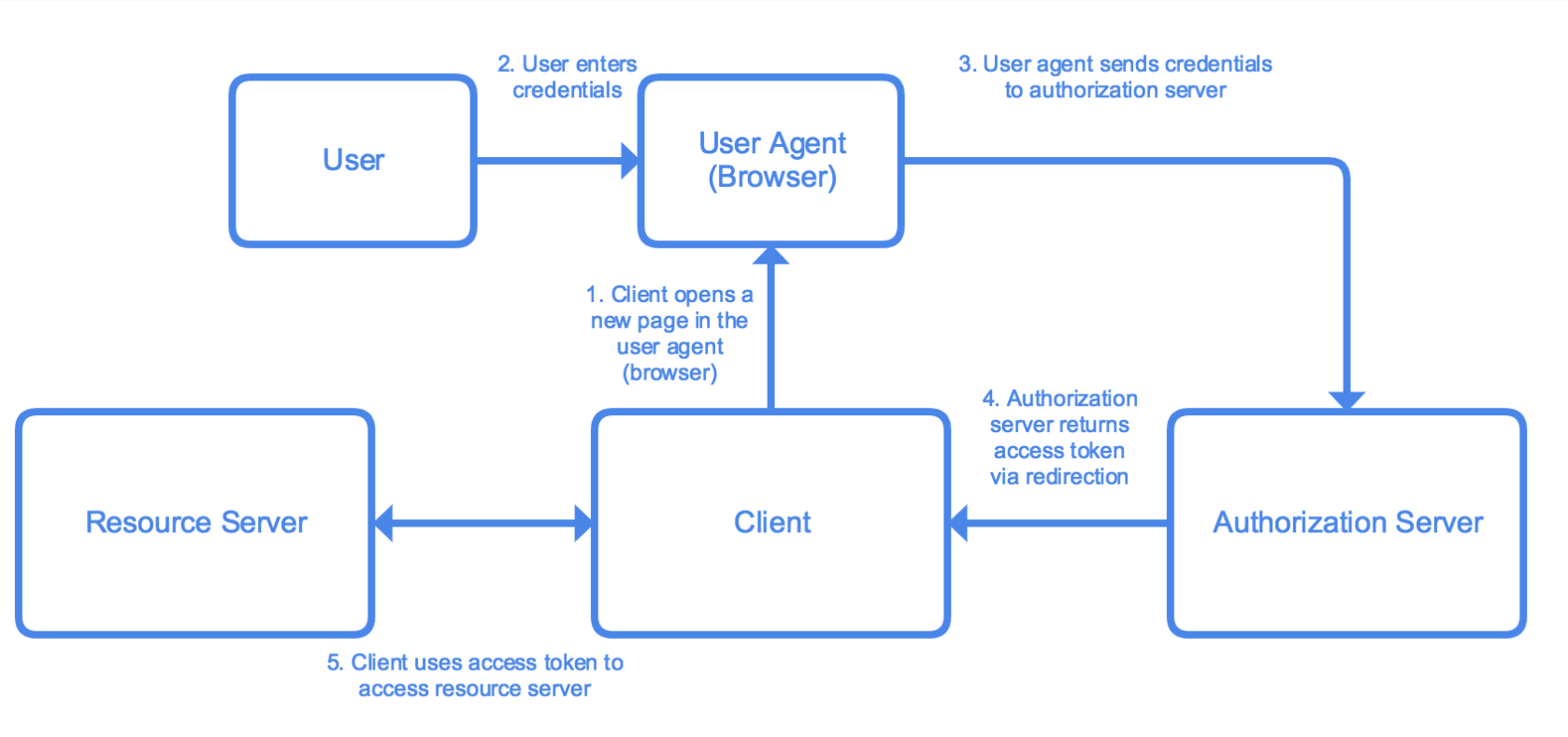
Implicit (1/2)

Fonctionnement

1. Le client commence par rediriger le propriétaire de la ressource vers le serveur d'autorisation. Le client doit inclure son identifiant dans la requête de redirection et le niveau d'accès qu'il souhaite obtenir.
2. Le propriétaire de la ressource s'authentifie auprès du serveur d'autorisation et approuve ou non la requête du client.
3. Si la requête est autorisée, le serveur d'autorisation redirige à nouveau le propriétaire de la ressource en utilisant l'URL de redirection défini par le client. La requête de redirection contient dans son URL le token d'accès qui permettra d'accéder aux ressources protégées.

Dans ce scénario le client n'est pas en mesure de conserver des infirmations d'autorisation de manière sûre.

Implicit (2/2)



source https://cloud.google.com/community/tutorials/understanding-oauth2-and-deploy-a-basic-auth-srv-to-cloud-functions

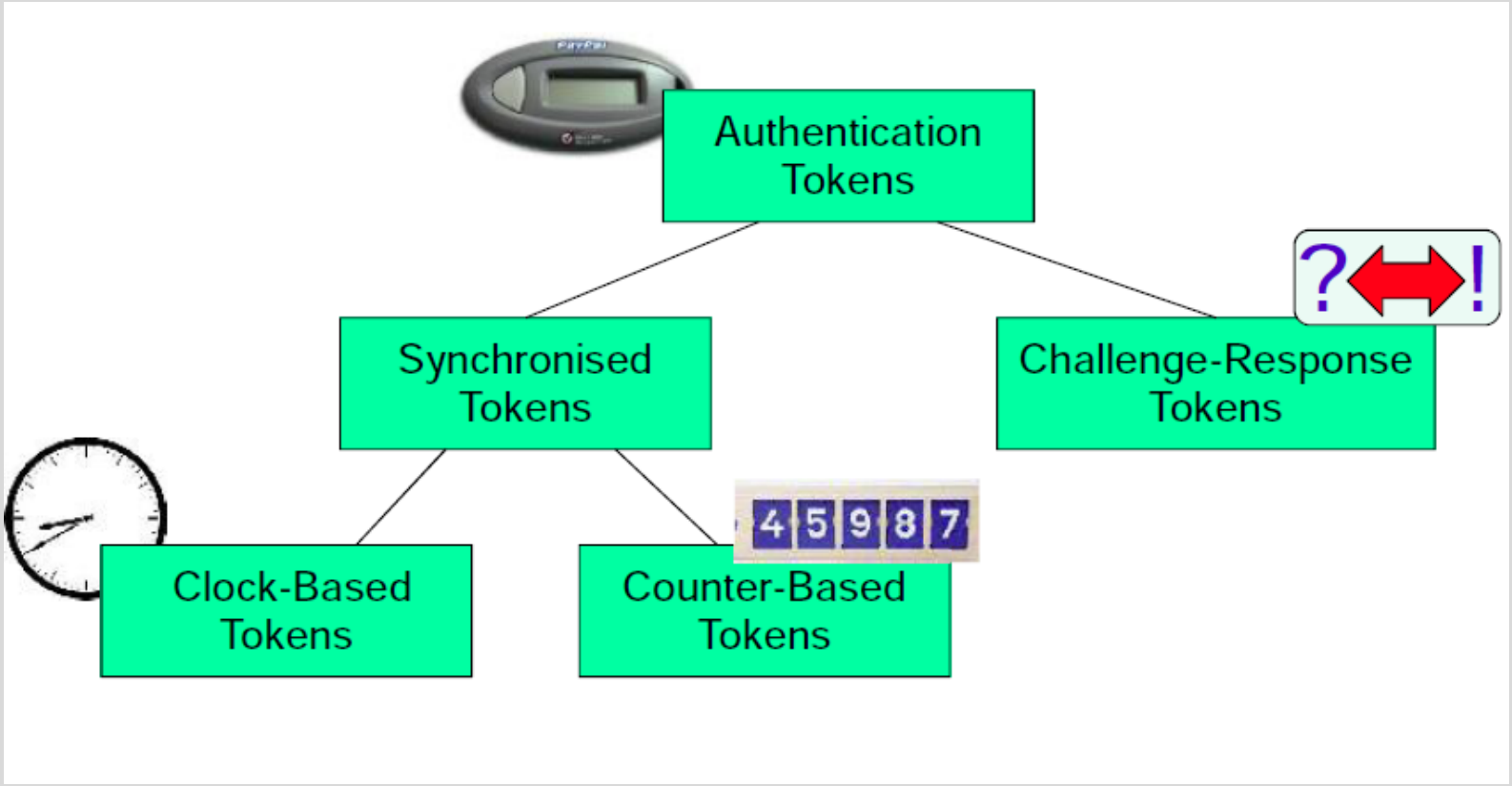
Plan

Identifiants pour l'authentification des utilisateurs

- Un identifiant est la "chose" utilisée pour l'authentification.
- On peut également l'appeler “token” ou “authentificateur”.
- par exemple, les mots de passe réutilisables, le code PIN, la biométrie, les cartes à puce, certificats, clés cryptographiques, tokens matériels OTP.
- Catégories d'identifiants:
  - Basé sur la connaissance (Quelque chose que vous connaissez) : Mots de passe
  - Basé sur la propriété (Quelque chose que vous avez) : Tokens
  - Basé sur l'inhérence (Quelque chose que vous êtes/faites) : Biométrie
- Authentification forte ou mulitfactorielle : Combinaison d'au moins deux types d'identifiants.

Indentifiants basés sur la propriété

Catégories de tokens



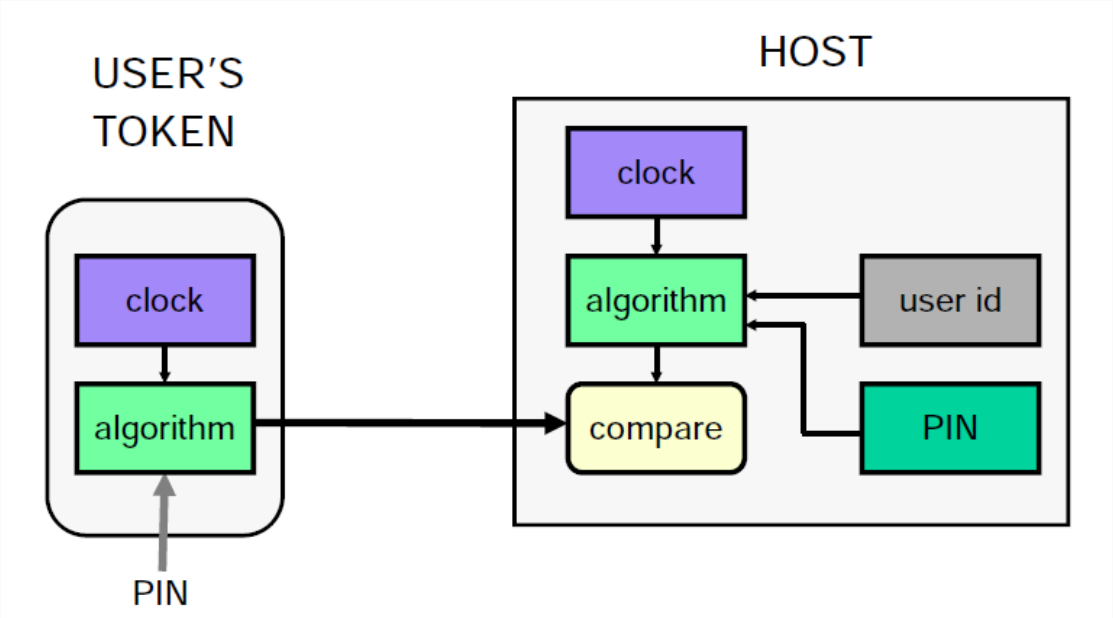
source https://www.uio.no/studier/emner/matnat/ifi/INF3510/v15/lectures/inf3510-2015-h08-user-authentication.pdf

OTP synchronisé (mot de passe unique)

- Utiliser un mot de passe une seule fois renforce de manière significative la sécurité de l'authentification des utilisateurs.
- Les générateurs de mots de passe synchronisés produisent la même séquence de mots de passe aléatoires au niveau token et du serveur d'authentification.
- Il existe deux méthodes générales :
  - Token basés sur le temps
  - Token basés sur un compteur

Token basés sur le temps

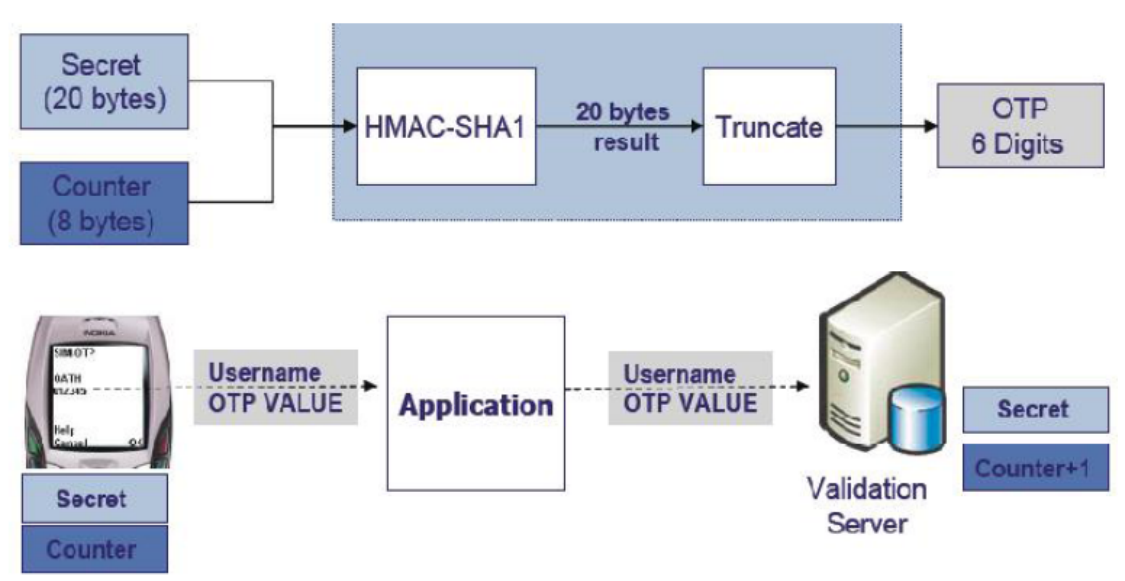
- Le token affiche un code dépendant du temps
- L'utilisateur copie le code du token pour se connecter
- La possession du token est nécessaire pour connaître le valeur correcte pour l'heure actuelle
- Chaque code est calculé pour une fenêtre temporelle spécifique
  - Les horloges doivent être synchronisées



source https://www.uio.no/studier/emner/matnat/ifi/INF3510/v15/lectures/inf3510-2015-h08-user-authentication.pdf

Token basés sur un compteur

- Les tokens basés sur des compteurs génèrent un "mot de passe".
  - Le mot de passe est le résultat de l'appel d'une fonction qui prend le compteur en argument ainsi que des données internes. Aucune donnée externe autre que le compteur n'est utilisée.
- Le HOTP est un mot de passe unique basé sur un algorithme de type HMAC décrit dans le RFC 4226



source https://www.uio.no/studier/emner/matnat/ifi/INF3510/v15/lectures/inf3510-2015-h08-user-authentication.pdf

Standards

OATH: Initiative for Open Authentication

Une collaboration industrielle pour développer une architecture de référence ouverte en tirant parti des normes ouvertes existantes pour la promotion de l'authentification forte.

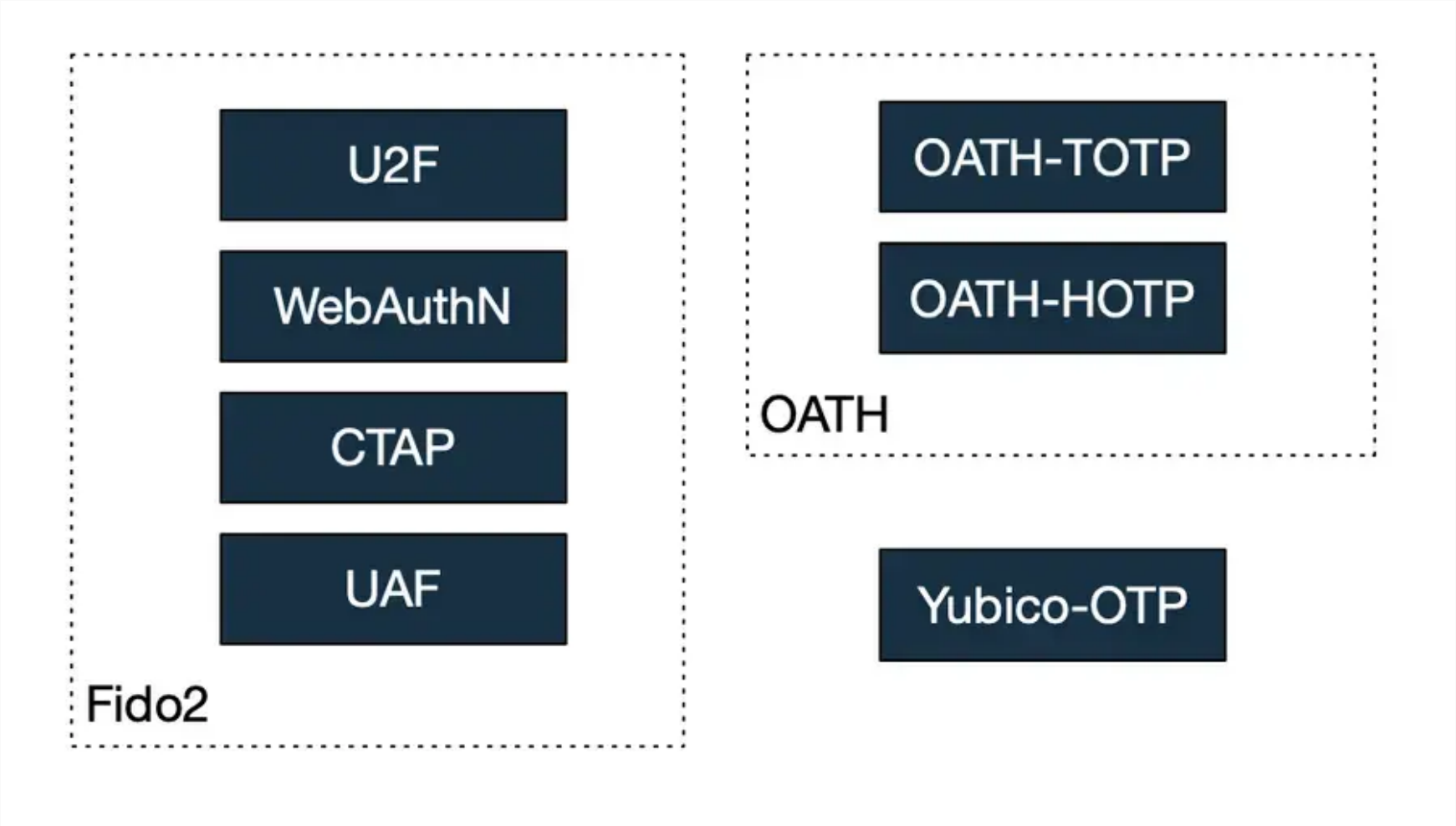
FIDO

Crée par Google et Yubico, *FIDO Alliance* a pour but de standardiser un protocole 2FA plus résistant: Universal 2nd Factor (U2F).

source https://fidoalliance.org/fido-specifications-technical-overview/

source https://2020.pass-the-salt.org/files/slides/PTS2020-Talk-02-FIDO2.pdf

Standards



source https://alicebobandeve.org/assets/article-images/fido-oath-overview-900-91cddd.webp

Passwords are a problem

"*71% of accounts are guarded by password used on multiple sites*" - **TeleSign**

"*62% of breaches involved the use of stolen credentials, brute force or phishing*" - **Verizon**

"*The vast majority of data breaches are caused by stolen or weak credentials*" - **Kaspersky**

"*86% of users would like to replace work-related password with fingerprint recognition technology if given the option*" – **Secret Double Octopus**

"*There is a consensus on the need to move away from passwords*" - **Forrester**

FIDO2

FIDO Alliance founded by:

Today, members also include:



- Developed by FIDO Alliance
  - FIDO = Fast IDentity Online
- 2 specifications
  - FIDO2 = WebAuthn + CTAP
- Addresses multiple authentication use cases
  - **Passwordless** (single factor)
  - **Multi factor** (passwordless + PIN or biometrics)
  - Second factor (CTAP1 / U2F)
    - Backwards compatible with U2F (Universal 2nd Factor) standard

Overview

Embedded or ejectable (USB/NFC)

Client device with web browser

Server at domain.com

Authenticator ↔ **Client** ↔ **Relying party** (RP)

CTAP2 API

WebAuthn API

Purpose of these 2 specifications

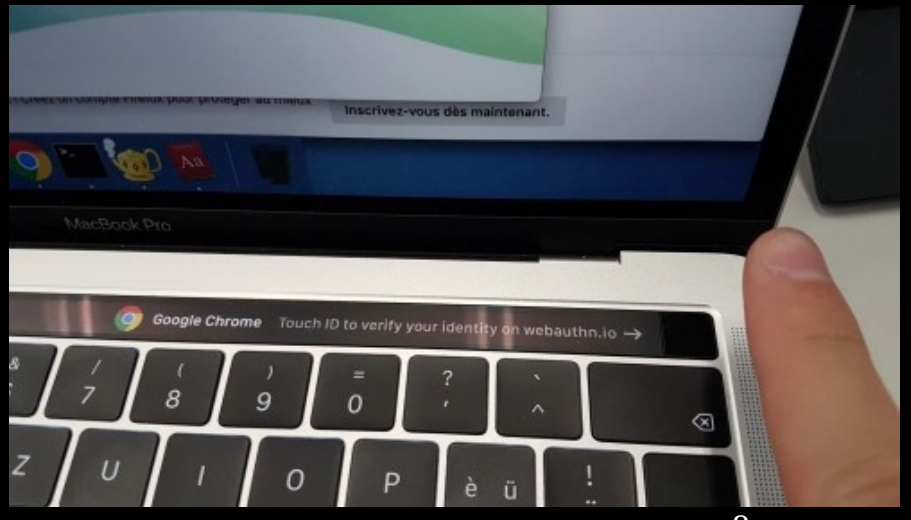
- **WebAuthn**
  - For web browsers
  - Javascript API
- **CTAP (Client To Authenticator Protocol)**
  - API between client and authenticator
    - Standard for all ejectable authenticators
  - Messages encoded in Concise Binary Object Representation (CBOR) format, RFC 7049
  - Also for desktop apps, command-line apps

Authenticators

- 2 authenticator types
  - **Platform authenticator** (Embedded/non-ejectable)
    - Your smartphone
    - Your laptop/desktop
  - **Roaming authenticator** (Ejectable)
    - A security key (USB or NFC)
    - Many vendors
      - Open source: Solo Key, see also: OpenSK
    - Entry price about $20 USD


iPhone


android
Version >= 7.0


Inscrivez-vous dès maintenant.
MacBook Pro
Google Chrome
Touch ID to verify your identity on webauthn.io

How does it work?

Registration

Registration

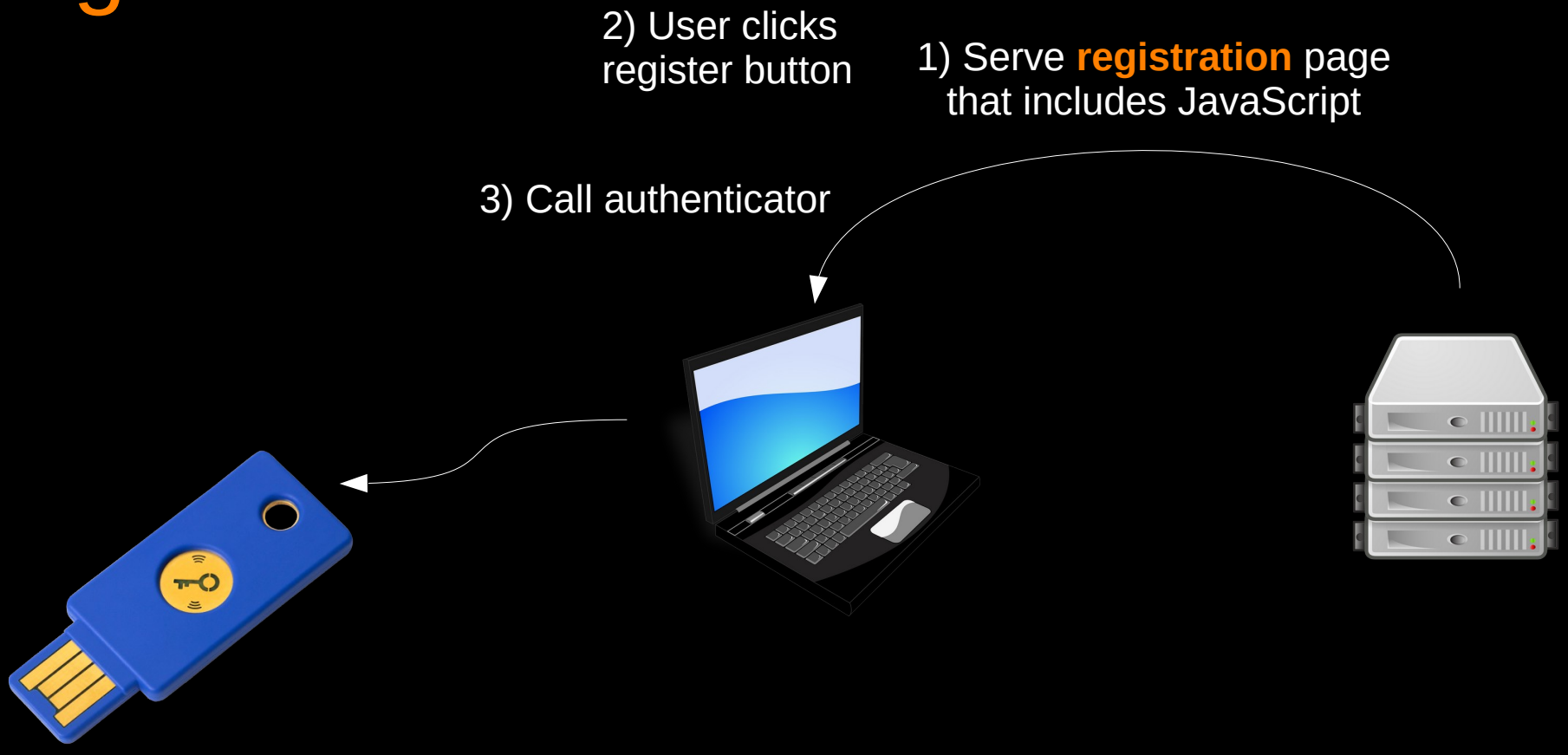
1) Serve **registration** page
that includes JavaScript

Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

Registration



Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

3) Call authenticator

4) User presence (UP) check,
User verification (UV) check (optional)

Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

3) Call authenticator

4) User presence (UP) check,
User verification (UV) check (optional)

5) Generate scoped key pair,
Store private key,
Return public key +
attestation signature

Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

3) Call authenticator

4) User presence (UP) check,
User verification (UV) check (optional)

5) Generate scoped key pair,
Store private key,
Return public key +
attestation signature

Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

3) Call authenticator

4) User presence (UP) check,
User verification (UV) check (optional)

6) Forward to RP

5) Generate scoped key pair,
Store private key,
Return public key +
attestation signature

Registration

2) User clicks register button

1) Serve **registration** page that includes JavaScript

3) Call authenticator

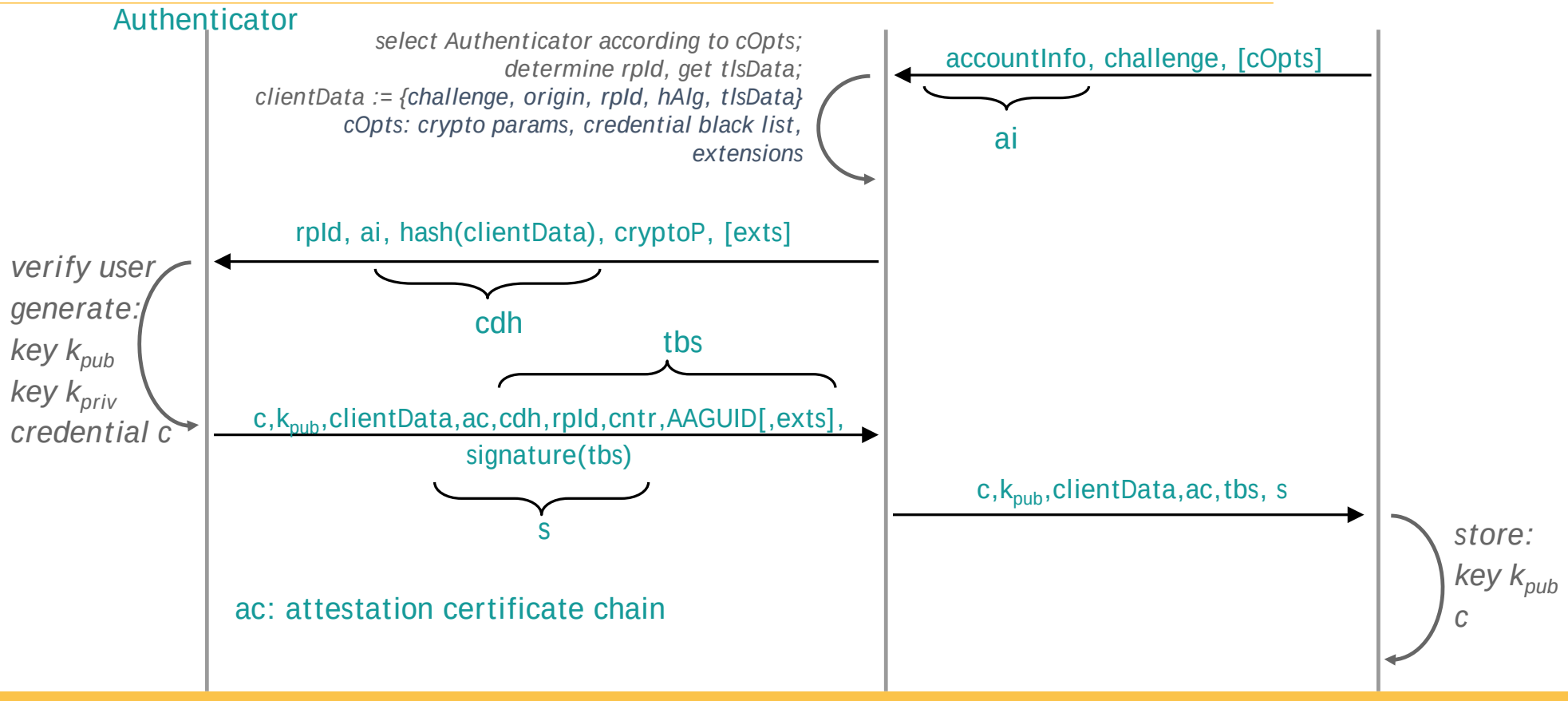
4) User presence (UP) check,
User verification (UV) check (optional)

6) Forward to RP

5) Generate scoped key pair,
Store private key,
Return public key +
attestation signature

7) Verify attestation,
Store public key

FIDO REGISTRATION



Authentication

Authentication

1) Serve **sign-in** page
that includes JavaScript

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

4) UP + UV checks

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

4) UP + UV checks

6) Forward to RP

5) Return **assertion** signature

Authentication

2) User clicks sign-in button

1) Serve **sign-in** page that includes JavaScript

3) Call authenticator

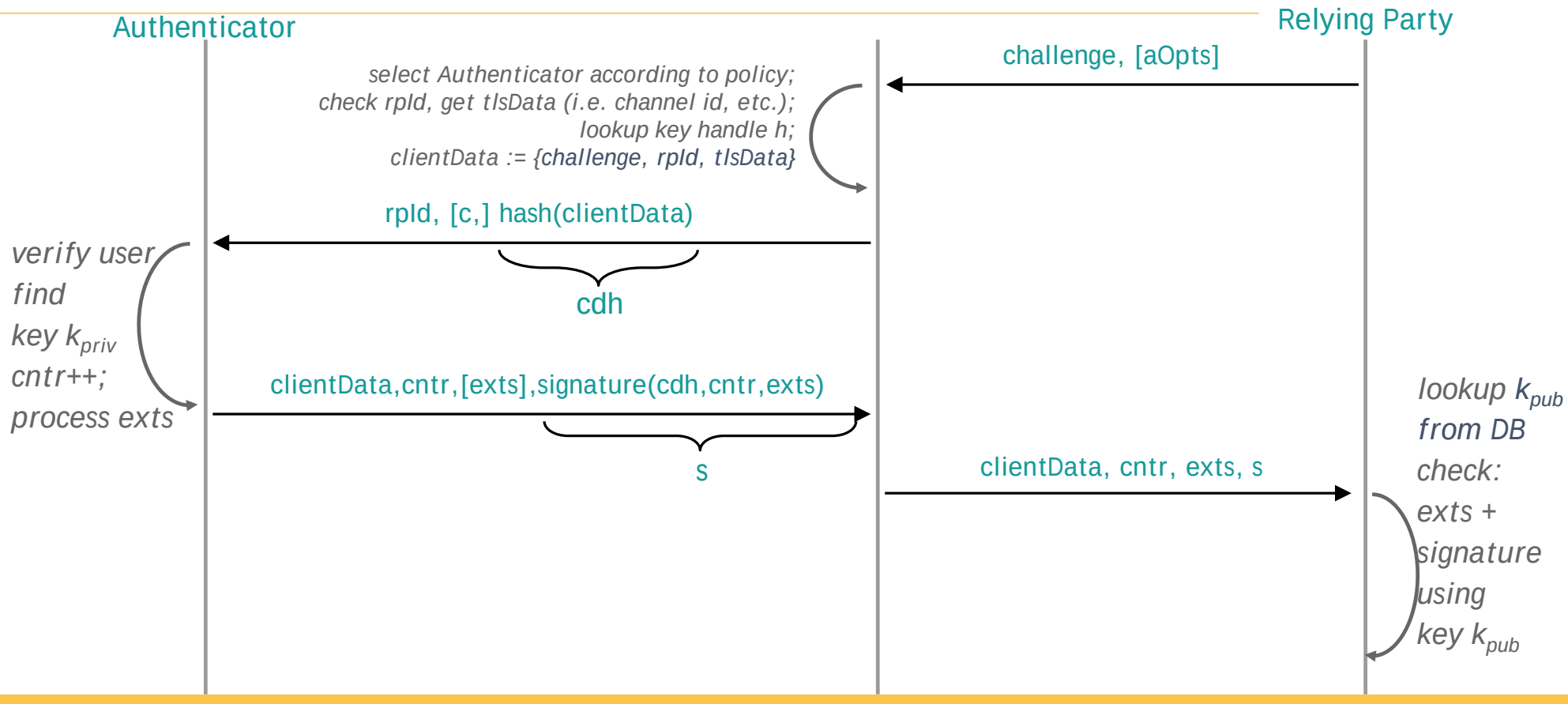
4) UP + UV checks

6) Forward to RP

5) Return **assertion** signature

7) **Verify assertion**, Authentication successful

FIDO AUTHENTICATION



Actor responsibilities

Authenticator main responsibilities

- **User presence** check
  - Tap authenticator
- **User verification** check (if supported)
  - PIN or biometrics
  - Yes, UV check is performed **client-side** (!)
- **Generate and store credentials**
- Produce signatures (**attestations** and **assertions**)

Client main responsibilities

- Act as **proxy** between authenticator and relying party
- Few other things
  - Example: if multiple accounts
    - Implement account selection logic

Relying party main responsibilities

- Verify attestations
- Verify assertions
- Check initial options (UV, ...)
- **Store public keys**
- Generate and verify challenges (prevent replay attack)
- Make authentication decision:
  - Authenticator characteristics and compromise status
  - Clone detection

Attestations

Why do we need attestations?

- RP can **trust authenticator is what it claims to be** by:
  - Verifying attestation signature using pre-established chain of trust
- If trusted, RP can:
  - Verify authenticator security level
  - Build an authenticator acceptance policy
  - Trust authenticity of authenticator data (including UV flag)

What is an attestation signature?

- Attestation is **optional** (!)
- Signature created during registration
- Signature is computed over:
  - **Authenticator data** (generated public key, AAGUID, UP, UV, etc.), and
  - Hash of **client data** (challenge, server origin, etc.)
- Multiple **attestation types**
  - Each attestation type provides a different trust model

Attestation types

- Basic attestation
- Self attestation
- Attestation CA (AttCA)
- ECDAA
- None

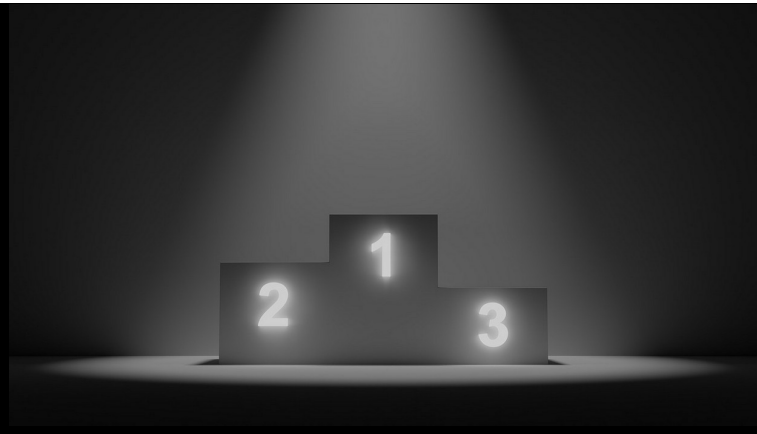
Basic attestation

- **Attestation private key** (burned in at factory)
  - Attestation certificate (contains public key)
  - Also certificate chain
- **Privacy vs compromise impact**: same attestation private key for ~100’000 authenticators of same model
  - Sweet spot for privacy and security
  - Ensure users cannot be tracked
  - Limit impact in case of attestation key compromise
- **Key compromise impact**
  - Cannot distinguish original authenticators and fake ones using leaked key
  - Authenticators registered before compromise are not impacted

Self attestation

- Generate key pair
- Sign using generated private key
  - Similar to self-signed certificates
- **Does not prove** that the authenticator is what it claims to be (!)
  - Only proves ownership of public key

Best attestation type?



- On paper, ECDAA for strict security policies
  - Banking, government
- ECDAA secure implementation is non-trivial
- Not every RP requires this security level
- In practice, may use **Basic attestation**, or not care about attestation at all
- Does not make a lot of sense to use complex attestation type with authenticators that do not provide strong protection against physical attacks

Assertions
(not attestations)

What is an assertion signature?

- Signature **created during sign-in**
- Produced using generated private key
- Is verified by RP using corresponding public key
- Also computed over:
  - Authenticator data
  - Hash of client data
- Many possible public key algorithms

Problem solved?

- No need to choose/remember/change passwords anymore
- Protocol prevents password re-use
- Invulnerable to phishing
- Strong protection against network attacks

FIDO2 best practices

- Make sure to **register a backup authenticator**
  - In case of physical theft, loss, your house burns, etc.
  - You won’t be locked out of your account if you have a backup method to sign-in
  - You can sign-in with the backup authenticator and revoke the stolen authenticator
- Set a PIN or biometric on your authenticator
  - The attacker still needs your PIN or fingerprint to sign-in

Password vs PIN

- “But you’re replacing the password with a PIN!”
- Password is sent over network and is vulnerable to all network attacks
- PIN is local
  - PIN does not need to be changed as often
- PIN cannot be brute forced
- Alternatively, use biometrics