Dites au revoir aux mots de passe avec WebAuthn !

William Petit

Avant de commencer

- S.C.O.P. dijonnaise de 14 personnes, depuis 2011
- Spécialisée dans le logiciel libre

Cadoles

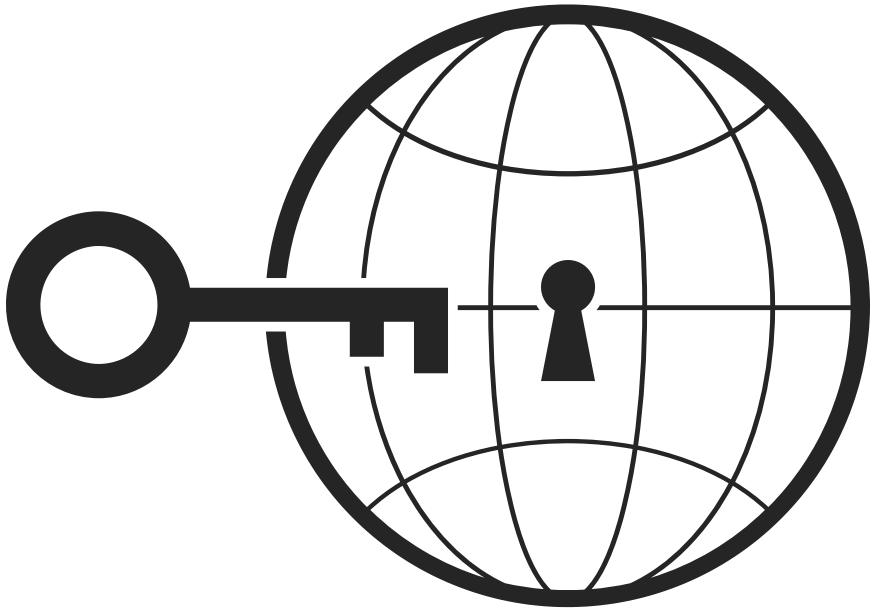
{{ La liberté est un choix }}

Un peu de contexte

- Je vous assure que c'est moi !
- Le mot de passe, cet ami qu'on aimerait voir moins souvent
- Dupon-d ou Dupon-t ?

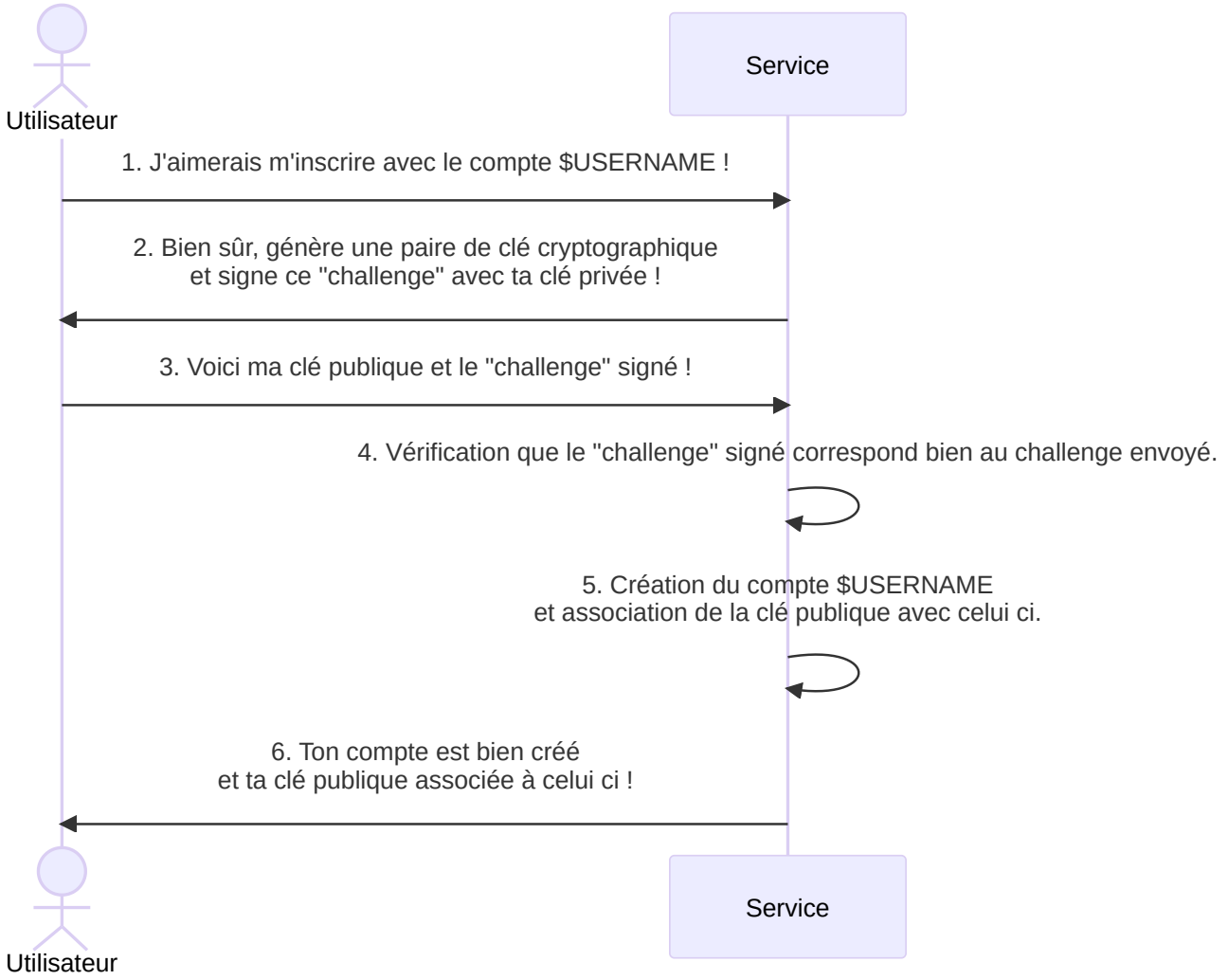
Qu'est ce que WebAuthn ?

- Une collaboration entre le W3C et l'alliance FIDO
- De l'authentification forte par paire de clés cryptographiques
- Authentification "sans mot de passe" ou vérification "double facteur"



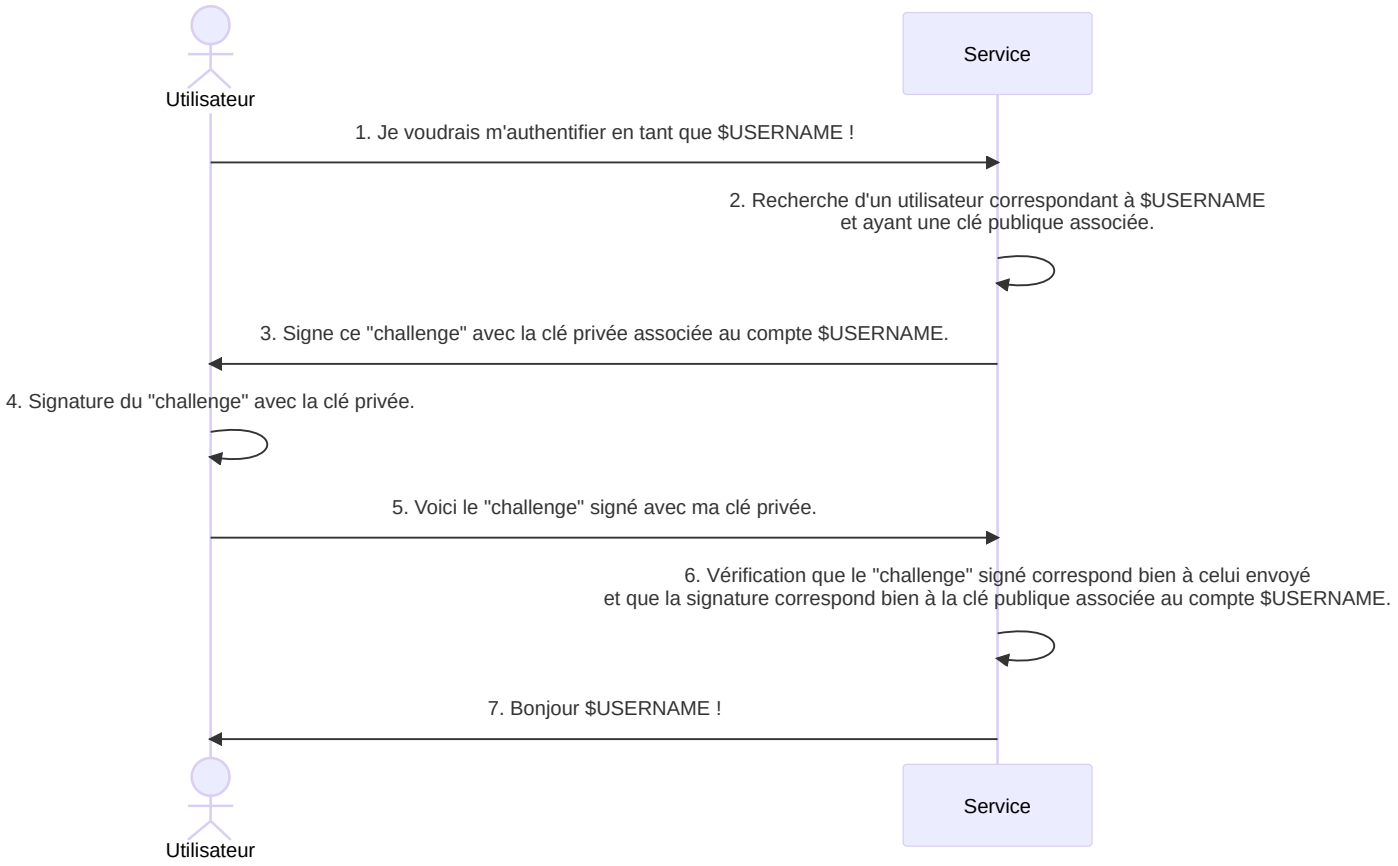
Authentification par paire de clés cryptographiques ? (1)

Inscription



Authentification par paire de clés cryptographiques ? (2)

Authentification



Passons à la technique

Grâce à la `Credential Management API` et notamment l'interface `PublicKeyCredential`.

Générer une accréditation ("credential")

```
// Transformation du "challenge" récupéré depuis
// le serveur
const challenge = Uint8Array.from(challengeFromServer, c => c.charCodeAt(0))

// Récupération de l'identifiant "opaque" généré par le serveur (<= 64 octets)
const userId = Uint8Array.from(userIdFromServer, c => c.charCodeAt(0))

const credentialOptions = {
    challenge,
    rp: { // "Relying Party"
        name: "Cadoles",    // Nom associé au RP
        id: "cadoles.com",  // Identifiant (domaine) associé au RP
    },
    user: {
        id: userId,     // Une séquence de données unique représentant l'utilisateur
        name: "jdoe",   // Nom d'utilisateur, spécifié (ou non) par le RP
        displayName: "John Doe", // Nom d'utilisateur (pour affichage)
    },
    pubKeyCredParams: [{alg: -7, type: "public-key"}], // Voir registre COSE, -7 = ES256
    authenticatorSelection: {
        authenticatorAttachment: "cross-platform", // Privilégier un module matériel (YubiKey) plutôt que lié à la plaforme (TouchID)
    },
    timeout: 60000,
    attestation: "direct" // On demande à recevoir les données directement générées par l'authentificateur
};

const credential = await navigator.credentials.create({
    publicKey: credentialOptions
});
```

Générer une affirmation ("assertion")

```
// On récupère le "challenge" à faire signer par le module d'authentification (envoyé normalement par le serveur)
const newChallenge = Uint8Array.from("myservernewchallenge", c => c.charCodeAt(0))

// On récupère l'identifiant de la clé associé à l'utilisateur (envoyé normalement par le serveur)
const keyRawId =  Uint8Array.from("myuserkeyid", c => c.charCodeAt(0))

const assertionOptions = {
    challenge: newChallenge,
    allowCredentials: [
        {
            id: keyRawId,
            type: 'public-key'
        }
    ],
    timeout: 60000
}

// On génère notre affirmation
navigator.credentials
  .get({ publicKey: assertionOptions })
```

Et le côté serveur alors ?

- Go - https://github.com/go-webauthn/webauthn
- TypeScript - https://github.com/passwordless-id/webauthn
- Ruby - https://github.com/cedarcode/webauthn-ruby

Quels pièges sur l'implémentation ?

Techniques

- Jongler entre les formats (`string` , `ArrayBuffer` , `Uint8Array` ...) et la sérialisation des données (`Base64` , `Base64URL`);
- Attention aux domaines (cf. `rp.id`);
- Automatisation des procédures de test encore complexe à ce jour.

UX

- La procédure de récupération de compte doit être pensée dès l'amorçage du projet pour pallier à la perte de l'authentificateur;

Quels facteurs de risque ?

- Ne pas essayer de ré-implémenter la partie serveur si vous pouvez utiliser une librairie maintenue par une communauté active (ou si vous êtes un véritable professionnel de la cryptographie);
- Pour limiter les risques de "lock-out", il faudrait pousser l'utilisateur à associer au minimum 2 authentificateurs avec son compte

Des questions ?

Bibliographie

- https://informationisbeautiful.net/visualizations/top-500-passwords-visualized/
- https://www.w3.org/TR/webauthn/
- https://webauthn.guide/
- https://fidoalliance.org/
- https://www.iana.org/assignments/cose/cose.xhtml