One Time Password

Mot de Passe à usage Unique

🔐 Définition et principe général

Technique d’authentification qui génère un mot de passe différent à chaque tentative de connexion.

Conçu pour être :

- **Unique** : Utilisé une seule fois et invalide dès qu’il est consommé.
- **Éphémère** : Sa durée de validité est limitée dans le temps (quelques dizaines de secondes à quelques minutes).
- **Non réutilisable** : Empêche les attaques par rejeu ou interception.

Permet de protéger les accès sensibles en rendant inutile toute tentative d’interception ou de récupération ultérieure du mot de passe, car celui-ci devient rapidement obsolète.

🎯 Pourquoi utiliser l’OTP en environnement technique ?

Aujourd’hui, sécuriser les applications web, les accès administratifs, ou les services sensibles avec un simple couple identifiant / mot de passe statique n’est plus suffisant. Les risques majeurs comprennent :

- **Interception des mots de passe** : Phishing, keyloggers, attaques MITM
- **Réutilisation frauduleuse** des informations d’identification compromises.
- **Faiblesse humaine** : mots de passe faibles, réutilisés ou faciles à deviner.

L’OTP répond directement à ces problématiques en introduisant une couche supplémentaire de sécurité dynamique, très efficace contre ces types d’attaques courantes.

⚙️ Types d’OTP courants : HOTP et TOTP

Deux grands standards techniques très utilisés :

- **HOTP (HMAC-based OTP)** : Basé sur un **compteur** partagé côté serveur et client. À chaque génération, le compteur s’incrémente, produisant un OTP différent.
  Exemples : Jetons physiques RSA SecurID, clés Yubikey
- **TOTP (Time-based OTP)** : basé sur le **temps**. Un secret partagé couplé à l’heure actuelle permet de générer un OTP valide pendant une courte fenêtre temporelle (généralement 30 secondes).
  Exemples : Google Authenticator, Microsoft Authenticator, Authy

📌 Comment fonctionne techniquement le TOTP

Deux éléments principaux :

- **Un secret partagé (clé secrète)** : Généralement généré côté serveur, puis distribué de manière sécurisée à l'utilisateur.
- **L'heure courante (UTC)** : utilisée pour générer des fenêtres de validité du mot de passe.

Client et Serveur génèrent indépendamment le même OTP, sans nécessiter de communication à chaque authentification. C'est une approche dite **stateless** très pratique pour gérer de nombreux utilisateurs simultanément.

🔐 TripAnArvor AuthentikATOR

Mise en œuvre d’un OTP

🧰 Environnement technique

Illustration concrète de la mise en place technique d’un OTP avec un environnement web typique constitué :

- **Back-end PHP** : Utilisation d’une librairie standardisée et open-source.
- **Front-end JS** : JavaScript natif (sans framework), afin de rester simple et proche des standards web modernes.

🧰 Environnement technique

Le backend PHP sera responsable des :

- **Secrets** : Générer les secrets OTP des utilisateurs.
- **Vérifications** : Vérifier les OTP fournis par les utilisateurs.

Le frontend JavaScript assurera :

- **Code OTP** : La collecte du code OTP depuis l'utilisateur.
- **Transmission** : La transmission sécurisée via HTTPS vers le serveur PHP pour validation.

🚨 Enjeux de sécurité techniques associés à OTP

Lors d’une mise en œuvre technique d’OTP, plusieurs points critiques doivent impérativement être maîtrisés :

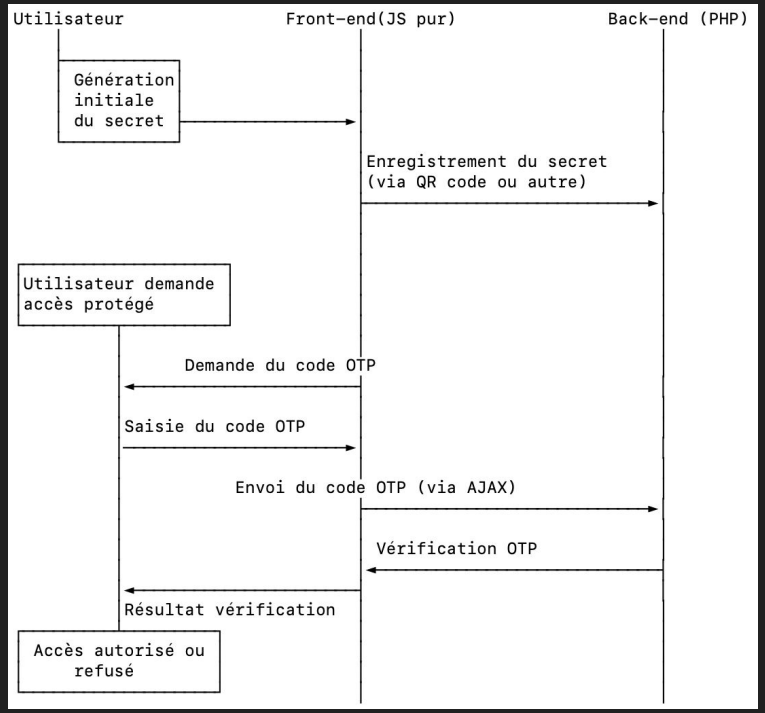
- **Confidentialité du secret partagé** :
  - Jamais exposé côté client (uniquement stocké côté serveur).
  - Protégé par chiffrement en base de données.
- **Synchronisation temporelle** : L’heure du serveur doit être synchronisée (NTP recommandé) pour éviter des erreurs de validation.

🚨 Enjeux de sécurité techniques associés à OTP

Lors d'une mise en œuvre technique d'OTP, plusieurs points critiques doivent impérativement être maîtrisés :

- **Protection contre la brute force** : Implémenter une limitation du nombre d'essais côté serveur pour éviter les attaques par force brute
- **Transmission sécurisée (HTTPS)** : Tout échange avec le serveur doit impérativement passer par un canal chiffré SSL/TLS.
- **Contrôle des erreurs** : Fournir le minimum d'informations lors d'échecs de validation pour éviter les fuites d'information.

🎯 Flux global du processus OTP



⚙️ Côté Backend : PHP

Utilisation d’une librairie PHP OTP (suggestion) :

- **Spomky Labs OTP** : https://github.com/Spomky-Labs/otphp

Fonctionnalités à prévoir :

- **Génération du secret partagé**
- **Sauvegarde du secret en BDD**
- **Vérification d’un code OTP envoyé depuis le client**

🌐 Côté Frontend : JavaScript natif (pur)

Le JS est utilisé simplement pour collecter le code OTP saisi par l'utilisateur et l'envoyer au serveur PHP via Ajax (fetch).

Fonctionnalités à prévoir :

- **Collecte d'un code**
- **Affichage d'un QRCode**

🌐 Côté Frontend : JavaScript natif (pur)

Note : vous coderez en JS/Ajax uniquement la partie mise en place de l'OTP (le 1er échange d'OTP qui entérine côté serveur l'activation de l'OTP), pour éviter une soumission de formulaire et avoir un retour OK/KO sans changement de page.

Ensuite, pour l'usage OTP au quotidien (le code OTP que l'utilisateur entre en même temps que son login et son mdp), vous conserverez le mécanisme de POST d'un formulaire avec donc un changement de page en retour de requête. Ceci afin de ne pas remettre en cause le mécanisme de login actuel sur vos sites.

🟢🟡🔵🔴 Les 4 phases du mécanisme OTP

🟢 Phase 1 : Génération initiale du secret OTP

Côté Serveur (PHP) :

- **Création du secret initial** : Le serveur génère un secret unique pour l'utilisateur (clé secrète).
- **Stockage sécurisé du secret** : Stockage du secret dans la base de données associée à l'utilisateur.
- **Transmission du secret à l'utilisateur** : Génération d'un QRCode ou affichage du secret à saisir dans une application telle que Google Authenticator.

🟡 Phase 2 : Configuration initiale utilisateur

Côté Client (Utilisateur) :

- **Installation d'une app OTP** : Google Authenticator, Authy, Microsoft Authenticator, 1Password, etc.
- **Ajout du secret à l'app OTP** : L'utilisateur scanne le QRCode ou saisit manuellement le secret communiqué par le serveur.
- **Renvoi du 1er code OTP** : Renvoie du 1er code OTP affiché dans l'app OTP pour valider, côté serveur, la bonne mise en place. C'est une phase non absolument nécessaire mais conseillée.

Après cette étape, le client et le serveur disposent du même secret partagé.

🔵 Phase 3 : Utilisation courante lors d’une connexion

Côté Client, lors d’une tentative d’authentification :

- **Génération automatique du code OTP par l’app client** : Toutes les ~30 secondes, un nouveau code (6 chiffres généralement) apparaît dans l’application. L’utilisateur copie ou retient ce code.
- **Saisie du code OTP dans l’interface d’authentification** : L’utilisateur saisit le code OTP actuellement visible dans l’app sur la page de login de votre site et le code est envoyé au serveur.

🔴 Phase 4 : Vérification du code OTP (serveur)

Côté Serveur (PHP). Lors de la réception d’une tentative d’authentification :

- **Réception du code OTP depuis le frontend** : Par soumission d’un formulaire ou via Ajax.
- **Vérification du code OTP avec la librairie** : Si valide autoriser la connexion, si invalide demander un nouveau code, refuser la connexion.

Livrables

🗒️ Livrables

Liste des 3 livrables attendus (la notation en tiendra compte) :

1️⃣ **Vidéo de présentation** : Dépôt sur un site de streaming ou de transfert de fichiers de votre choix. Vous devez fournir une URL publique d'accès direct à cette vidéo dans un fichier **VIDEO.md**

2️⃣ **Codes PHP et JS** : Une archive **tgz** ou **zip** (et rien d'autre) de tout le code source concerné par OTP, déposé sur Moodle par un seul membre de l'équipe. Le fichier doit porter le numéro de votre équipe (ex : **A12.zip**, **B22.tgz**).

3️⃣ **Documentation** : un **PDF** décrivant la mise en œuvre technique de votre OTP.