SAE Cryptographie

Rapport d'Analyse

Module R3.09

Cryptographie et Sécurité

UPJV – IUT d'Amiens

Département Informatique

BUT 2ème année

Réalisé par :

Clément SCHILLER - Léanne BASIN - Loïc RESTOUT - Louka CARPENTIER - Maxence LEPEUVE - Noa ARNOULD

Date : 6 décembre 2025

Table des matières

1 Stockage des mots de passe

Le stockage sécurisé des mots de passe est un pilier fondamental de la sécurité informatique. Voici les quatre méthodes principales, de la moins sécurisée à la plus robuste.

1.1 A. Stockage en clair

Danger Critique

Définition : Le mot de passe est enregistré tel quel dans la base de données (ex : "Azerty123").
Attaque : Si un pirate accède à la base de données, il **lit simplement** tous les mots de passe. Accès immédiat et total.

1.2 B. Mots de passe Hachés (Hash)

Définition : Le hachage transforme le mot de passe en une empreinte numérique fixe (ex : SHA-256) via un algorithme à sens unique.
Mise en place : `bdd_password = hash(user_password)`

Vulnérabilité

L'attaquant utilise des **Rainbow Tables** (tables pré-calculées) pour retrouver instantanément les mots de passe courants à partir de leur hash.

1.3 C. Mots de passe Salés (Salt + Hash)

Définition : On ajoute une chaîne aléatoire unique (le "sel") à chaque mot de passe avant de le hacher. Le sel est stocké avec le hash.
Mise en place : `bdd_password = hash(user_password + salt)`
Impact sur l'attaquant : Les Rainbow Tables deviennent inutiles. L'attaquant doit faire une attaque par force brute **spécifique pour chaque utilisateur**, ce qui est extrêmement coûteux en temps.

1.4 D. Mots de passe Poivrés (Pepper + Salt + Hash)

Recommandation de Sécurité Maximale

Définition : Le poivre est une clé secrète stockée **en dehors de la base de données** (ex : HSM, variables d'env).
Mise en place : `bdd_password = hash(user_password + salt + pepper)`
Impact : Même si la base de données est compromise (Injection SQL), l'attaquant ne peut pas "cracker" les mots de passe car il lui manque le poivre.

2 Algorithme SHA-256

2.1 Fonctionnement

SHA-256 (Secure Hash Algorithm 256-bit) produit une empreinte unique de 256 bits.

1. **Padding :** Le message est complété pour être un multiple de 512 bits.
2. **Découpage :** Le message est traité par blocs de 512 bits.
3. **Compression :** 64 tours de calculs (opérations logiques bit-à-bit, rotations, additions) mélangent les données.
4. **Résultat :** Une chaîne hexadécimale de 64 caractères.

Effet Avalanche

Une modification infime de l'entrée (un seul bit) change complètement le hash de sortie ($\approx 50\%$ des bits changent). Cela garantit l'imprédictibilité.

3 Robustesse des mots de passe

3.1 Court/Complexe vs Long/Simple

Réponse : Il vaut mieux un **mot de passe long** (alphanumérique) qu'un mot de passe court (avec caractères spéciaux).

3.2 Explication par l'entropie (N^L)

— **Court Complexe :** 8 caractères, 90 symboles.

$$E = 90^8 \approx 4 \times 10^{15} combinaisons$$

— **Long Simple :** 16 caractères, 36 symboles.

$$E = 36^{16} \approx 7 \times 10^{24} combinaisons$$

Conclusion : La longueur (L en exposant) influence beaucoup plus la sécurité que la complexité (N en base). Une "phrase de passe" est donc préférable.

4 TLS et Authentification Client

4.1 Authentification mutuelle (mTLS)

Dans un échange TLS classique, seul le serveur présente un certificat. En **mTLS**, le serveur demande aussi au client de prouver son identité via un certificat client.

Avantages : Sécurité maximale (preuve de possession de clé privée), protection contre le phishing.

Inconvénients : Gestion lourde des certificats (PKI), complexité de déploiement pour l'utilisateur final.

4.2 Cas d'usage obligatoire

Le mTLS est recommandé pour les architectures **Zero Trust** :

— Communications **Server-to-Server** (API internes).

— Accès **VPN** d'entreprise.

— Appareils **IoT** (capteurs critiques).

5 Authentification : Token vs Cookie

5.1 Comparaison

Critère	Cookie de Session (Stateful)	Token JWT (Stateless)
Stockage Serveur	L'ID de session est stocké en RAM/BDD.	Rien n'est stocké (tout est dans le token).
Scalabilité	Difficile (nécessite synchro type Redis).	Excellente (vérification locale par signature).
Révocation	Facile (suppression côté serveur).	Difficile (nécessite Blacklist ou expiration courte).
Format	Chaîne opaque aléatoire.	JSON encodé (Header.Payload.Signature).

5.2 Fonctionnement du JWT

Un JSON Web Token contient les données de l'utilisateur (Payload) et est signé cryptographiquement par le serveur.

Utilisation préférée

Les JWT sont le standard pour les **API REST**, les **Applications Mobiles** et les **Single Page Apps (SPA)**, car ils facilitent l'architecture sans état (Stateless).