

Sujet 1 - HelpHub

S3.01 : Rapport de test de sécurité

ANNEE 2023/2024

22/12/2024

*Auteurs :*

EVRARD Thibaud

GALLOTTA Maxime

MARTIN Matéo

RABEHI Milhane

**Sommaire :**

[1. Chiffrement 2](#_Toc155871380)

[1.a) Chiffrement 2](#_Toc155871381)

[1.b) Déchiffrement 2](#_Toc155871382)

[1.c) Clé secrète 2](#_Toc155871383)

[1.d) La méthode de chiffrement 2](#_Toc155871384)

[1.e) Fonctionnement de ‘AES-256-CBC’ 3](#_Toc155871385)

[1.f) Données cryptées : 3](#_Toc155871386)

[2. Hachage 3](#_Toc155871387)

[2.a) Hachage du mot de passe 3](#_Toc155871388)

[2.b) Authentification de l'Utilisateur 3](#_Toc155871389)

[2.c) La méthode de hachage 4](#_Toc155871390)

[2.d) Fonctionnement de ‘bcrypt’ 4](#_Toc155871391)

Que ça soit pour chiffrer ou crypter les données nous avons décidé d’utiliser les fonctions fournies par Laravel. Nous avons fait ce choix avant tout pour des raisons de simplicité, en effet nous n’avions pas beaucoup de temps pour faire ce projet et donc ne voulions pas perdre de temps à importer d’autres fonctions ou encore moins de créer nous-mêmes les algorithmes. De plus Laravel est un outil professionnel auquel nous pouvons nous fier et qui utilise des techniques de hachage et cryptage efficace et sécurisé.

# Chiffrement

Pour le chiffrement et le déchiffrement des données nous avons utilisé les fonctions *encrypt* et *decrypt* fournies par le service de chiffrement intégré à Laravel. Ces fonctions sont utilisées pour chiffrer et déchiffrer des données sensibles, comme des informations bancaires, avant de les stocker dans la base de données.

#### Chiffrement

x²

Il suffit d’utiliser *Crypt ::encrypt* pour chiffrer la valeur (ici le numéro de compte), avant de la stocker dans la base de données. Cela garanti que les informations sensibles ne sont pas stockées en brut. La valeur ressemblera à ça en base de données :



#### Déchiffrement



Pour récupérer les données cryptées, afin par exemple de les afficher en clair sur le site, on utilise *Crypt ::decrypt*.

#### Clé secrète

Le chiffrement et le déchiffrement nécessitent une clé secrète, qui est configurée dans le fichier de configuration de Laravel (config/app.php) comme ceci :



Cette ligne spécifie la clé de chiffrement utilisée par Laravel. La clé est essentielle pour le fonctionnement du chiffrement et du déchiffrement. Elle doit être unique et secrète. Laravel utilise une clé générée aléatoirement pour chaque application lors de l'installation. La clé est stockée dans la variable d'environnement APP\_KEY. Cette variable APP\_KEY est configuré dans le fichier .env du projet comme ceci :



#### La méthode de chiffrement

La méthode de chiffrement utilisé est configurée dans le fichier de configuration de Laravel (config/app.php) comme ceci :



Cette ligne indique l'algorithme de chiffrement utilisé par Laravel. Dans notre cas, l'algorithme est AES (Advanced Encryption Standard) avec une longueur de clé de 256 bits en mode de chiffrement CBC (Cipher Block Chaining). CBC est un mode de chiffrement par bloc qui ajoute un niveau supplémentaire de sécurité en utilisant un vecteur d'initialisation pour chaque bloc de données.

#### Fonctionnement de ‘AES-256-CBC’

**1ère étape :** Diviser la chaîne en blocs de données

L'algorithme AES travaille par blocs de données. La longueur du bloc dépend de la taille de la clé et du mode de chiffrement (dans le cas de CBC, c'est généralement 128 bits ou 16 octets).

**2ème étape :** Initialisation du vecteur d'initialisation (IV)

Pour le mode CBC, un vecteur d'initialisation est utilisé pour ajouter de l'aléatoire aux premiers blocs chiffrés. Le IV doit être différent pour chaque exécution du chiffrement pour éviter certaines attaques.

**3ème étape :** Chiffrement de chaque bloc

L'algorithme AES chiffre chaque bloc de données avec la clé secrète. Pour le premier bloc, le IV est « XORé » avec le bloc de données avant d'être chiffré. Pour les blocs suivants, le bloc chiffré précédent est XORé avec le bloc de données actuel avant le chiffrement. Cela introduit une dépendance entre les blocs et rend le chiffrement en mode CBC plus sûr.

**4ème étape :** Création d'un texte chiffré

Les blocs chiffrés sont concaténés pour former le texte chiffré final.

Le processus de déchiffrement est très similaire au chiffrement. On commence par diviser la valeur en bloc comme pour le chiffrement. On déchiffre ensuite chaque bloc à l’aide de la clé secrète. Chaque bloc est XORé avec le bloc précédent. Et enfin, chaque bloc est concaténé pour retrouver la valeur originale.

#### Données cryptées :

* Numéro de carte bancaire des utilisateurs
* Date d’expiration de carte bancaire des utilisateurs
* Cryptogramme de carte bancaire des utilisateurs
* Nom de carte bancaire des utilisateurs
* Rib des associations pour les dons
* Numéro de compte bancaire des utilisateurs
* Nom de compte bancaire des utilisateurs

# Hachage

Le hachage est un processus unidirectionnel qui transforme les données en une empreinte numérique unique, appelée "hash". Une caractéristique clé du hachage est qu'il est difficile, voire impossible, de retrouver les données d'origine à partir de la valeur hachée. C'est pourquoi nous avons décidé de hacher les mots de passe avant de les stockés plutôt que de le crypté. D'un autre côté, le cryptage est bidirectionnel, ce qui signifie que les données peuvent être transformées de manière réversible à l'aide d'une clé. L'utilisation du cryptage pour stocker les mots de passe pourrait potentiellement permettre à un attaquant d'inverser le processus et récupérer les mots de passe d'origine.

#### Hachage du mot de passe

Le mot de passe des utilisateurs est la seule valeur hachée dans notre projet.



Nous utilisons *Hash::make* pour hacher le mot de passe avant de le stocker dans la base de données. Un fois stocké, le mot de passe est une suite de caractère sans sens comme ceci :



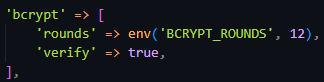
#### Authentification de l'Utilisateur

****

Cette ligne est utilisée pour tenter d'authentifier l'utilisateur en vérifiant les informations d'identification par rapport à celles stockées dans la base de données. Le programme va chiffrer le mot de passe entré par le visiteur pour voir si le hash correspond bien au hash en base de données à la ligne de l’adresse mail correspondante à celle rentré par l’utilisateur.

#### La méthode de hachage

Laravel utilise la bibliothèque Bcrypt (Blowfish Crypt) pour le hachage des mots de passe. Le hachage Bcrypt est une fonction de hachage de mot de passe conçu pour être lent, ce qui rend difficile l'utilisation d'attaques par force brute ce qui le rend résistant aux attaques par force brute. Dans Laravel, la configuration liée au hachage des mots de passe est dans le fichier config/hashing.php comme ceci :



'rounds' => env('BCRYPT\_ROUNDS', 12), : Cette ligne spécifie le nombre de tours que l'algorithme Bcrypt effectuera lors du hachage des mots de passe. Ici le nombre de tours est configuré en utilisant la fonction env, qui récupère la valeur de la variable d'environnement BCRYPT\_ROUNDS que nous n’avons pas définie, elle utilise donc la valeur par défaut de 12. Augmenter le nombre de tours rend le hachage plus sécurisé, mais cela peut également ralentir le processus. Tandis que réduire le nombre de tour pourrait réduire la sécurité.

'verify' => true, : Cette ligne indique que le système doit vérifier automatiquement si le mot de passe doit être rehashé à chaque fois qu'un utilisateur se connecte. Cela augmente encore plus la sécurité et permet aussi d’utiliser la dernière version de cryptage si celle-ci est mise à jour.

#### Fonctionnement de ‘bcrypt’

**1ère étape :** Récupération du Mot de Passe

Lorsqu'un utilisateur fournit un mot de passe lors de l'inscription ou de la mise à jour de son mot de passe, Laravel récupère cette chaîne de caractères.

**2ème étape :** Génération d'un Salt Aléatoire

Bcrypt génère un Salt aléatoire afin que deux même mot de passe n’est pas le même hash.

**3ème étape :** Combinaison du Mot de Passe et du Sel

Le mot de passe fourni par l'utilisateur est combiné avec le sel généré aléatoirement. Cela crée une chaîne de caractères plus longue et complexe qui sera effectivement hachée.

**4ème étape :** Application de l'Algorithme Bcrypt :

L'algorithme Bcrypt est appliqué à la chaîne résultante (mot de passe + sel) avec le nombre de tours spécifié. Une fois tous les tours terminés la chaîne est renvoyer et constitue le hash du mot de passe.