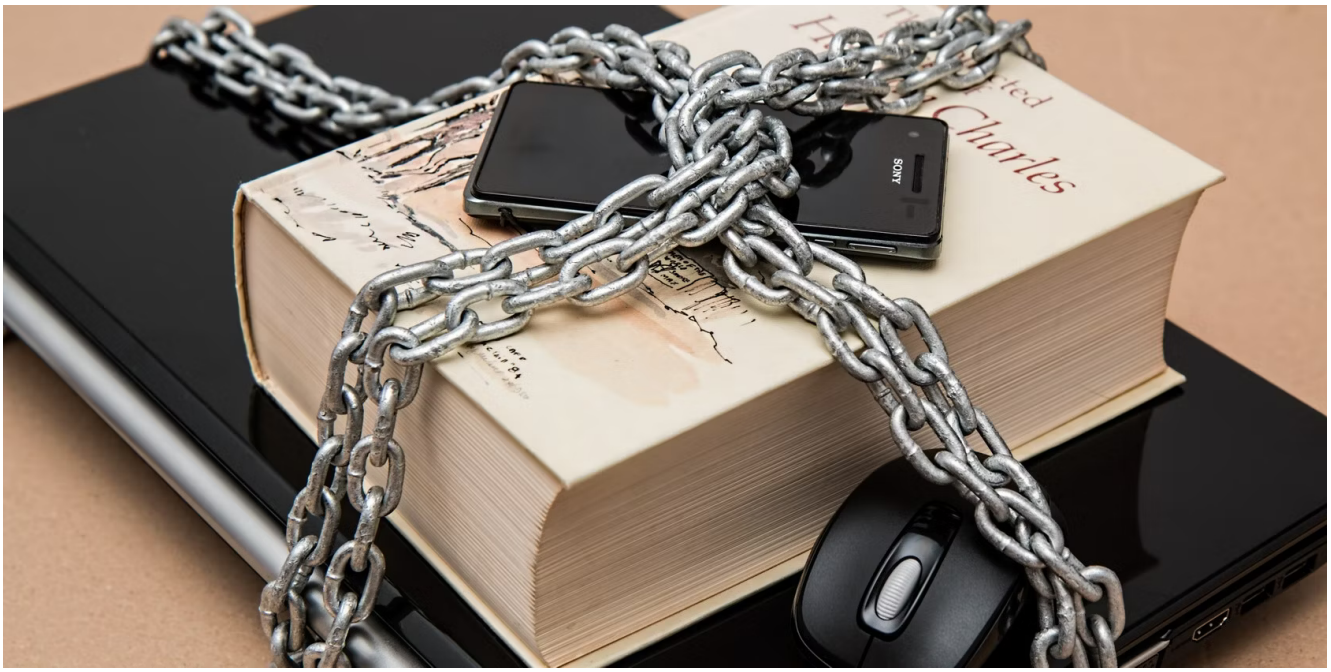
Comment hacher et vérifier un mot de passe en Node.js avec bcrypt

Par **Sandrine Cestpafo** - 13 avril 2023 Date modifiée: 13 avril 2023



L'une des meilleures façons de stocker les mots de passe en toute sécurité est de les saler et de les hacher. Le salage et le hachage convertissent un mot de passe simple en une valeur unique difficile à inverser. La bibliothèque Bcrypt vous permet de hacher et de saler les mots de passe dans Node.js avec très

peu d’efforts.

Qu’est-ce que le hachage de mot de passe ?

Détails du contenu : [masquer]

Le hachage de mot de passe consiste à faire passer un mot de passe en texte clair par un algorithme de hachage afin de générer une valeur unique. Cette valeur unique est appelée hachage. Quelques exemples d’algorithmes de hachage sont bcrypt, scrypt et SHA.

L’une des principales propriétés d’un bon algorithme de hachage est qu’il génère la même sortie pour la même entrée. Cette prévisibilité rend les hachages vulnérables aux attaques par force brute. Un pirate peut calculer à l’avance les valeurs de hachage pour de nombreuses entrées couramment utilisées, puis les comparer aux valeurs de hachage des valeurs cibles. Vous pouvez atténuer cette vulnérabilité en utilisant le salage.

Qu’est-ce que le salage des mots de passe ?

Le salage des mots de passe consiste à ajouter une chaîne aléatoire (le sel) à un mot de passe avant de le hacher. De cette manière, le hachage généré

sera toujours différent à chaque fois. Même si un pirate obtient le mot de passe haché, il lui faudra beaucoup de temps pour découvrir le mot de passe original qui l'a généré.

Comment utiliser Bcrypt pour hacher et vérifier un mot de passe

bcrypt est un module npm qui simplifie le hachage des mots de passe dans Node.js. Pour l'utiliser, suivez les étapes ci-dessous :

Etape 1 : Installer Bcrypt

Installez bcrypt en exécutant les commandes suivantes sur le terminal.

Utilisation de npm :

```
npm install bcrypt
```

Utilisation de yarn :

```
yarn add bcrypt
```

Étape 2 : Importer Bcrypt

Au début de votre fichier JavaScript, importez Bcrypt.

```
const bcrypt = require("bcrypt")
```

Étape 3 : Générer un sel

Appeler le **bcrypt.genSalt()** pour générer un sel. Cette méthode accepte une valeur entière qui est le facteur de coût déterminant le temps nécessaire pour hacher un mot de passe. Plus le facteur de coût est élevé, plus l'algorithme prend du temps et plus il est difficile d'inverser le mot de passe crypté en utilisant la force brute.

Une bonne valeur doit être suffisamment élevée pour sécuriser le mot de passe, mais aussi suffisamment basse pour ne pas ralentir le processus. Elle est généralement comprise entre 5 et 15. Dans ce tutoriel, nous utiliserons 10.

```
bcrypt.genSalt(10, (err, salt) => {
  // use salt to hash password
})
```

Étape 4 : hachage du mot de passe

Dans la fenêtre **bcrypt.genSalt** transmet le mot de passe en clair et le sel généré à la fonction **bcrypt.hash()** méthode pour hacher le mot de passe.

```
bcrypt.genSalt(10, (err, salt) => {
  bcrypt.hash(plaintextPassword, salt, function(err, hash) {
      // Store hash in the database
  });
})
```

Une fois le hachage généré, stockez-le dans la base de données. Vous l'utiliserez pour vérifier un mot de passe et authentifier un utilisateur qui tente de se connecter.

Au lieu de générer le sel et le hachage séparément, vous pouvez également les générer automatiquement à l'aide d'une seule fonction.

```
  bcrypt.hash(plaintextPassword, 10, function(err, hash) {
    // store hash in the database
});
```

Étape 5 : Comparer les mots de passe à l'aide de bcrypt

Pour authentifier les utilisateurs, vous devez comparer le mot de passe qu'ils fournissent avec celui de la base de données à l'aide de la fonction **bcrypt.compare()** fonction. Cette fonction accepte le mot de passe en texte clair et le hachage que vous avez stocké, ainsi qu'une fonction de rappel. Cette fonction de rappel fournit un objet contenant toutes les erreurs qui se sont produites et le résultat global de la comparaison. Si le mot de passe correspond au hachage, le résultat est vrai.

```
  bcrypt.compare(plaintextPassword, hash, function(err, result) {
    if (result) {
        // password is valid
    }
});
```

Utiliser Async/Await

Vous pouvez crypter des mots de passe dans Node.js avec Bcrypt en utilisant async/await comme suit.

```
  async function hashPassword(plaintextPassword) {
    const hash = await bcrypt.hash(plaintextPassword, 10);
    // Store hash in the database
}

// compare password
async function comparePassword(plaintextPassword, hash) {
    const result = await bcrypt.compare(plaintextPassword, hash);
    return result;
}
```

Utiliser les promesses

La bibliothèque bcrypt supporte également l’utilisation de promesses. Par exemple, voici une fonction qui hash le mot de passe en utilisant le bloc then…catch.

```
  function hashPassword(plaintextPassword) {
    bcrypt.hash(plaintextPassword, 10)
        .then(hash => {
            // Store hash in the database
        })
        .catch(err => {
            console.log(err)
        })
}
```

De même, cette fonction compare un mot de passe en clair de l’utilisateur à un mot de passe haché en utilisant des promesses.

```
function comparePassword(plaintextPassword, hash) {
   bcrypt.compare(plaintextPassword, hash)
       .then(result => {
           return result
       })
       .catch(err => {
           console.log(err)
       })
}
```

Le hachage et le salage sont une victoire facile

Vous pouvez utiliser la bibliothèque Bcrypt pour hacher et vérifier les mots de passe dans Node.js. Le hachage des mots de passe minimise les risques que des cybercriminels accèdent à des mots de passe simples et les utilisent pour accéder à des données ou des services sensibles.

Le salage de vos mots de passe hachés les rend encore plus sûrs. Outre le hachage, il convient de toujours valider la force du mot de passe, ce qui constitue une mesure de sécurité supplémentaire.