TP 6 : Stockage de mots de passe

L'objectif de ce TP est de vous faire découvrir diverses façons, plus ou moins sécurisées, de stocker des mots de passe dans une base de données d'utilisateurs.

Exercice 1 (Stockage en clair des logins et des mots de passe d'utilisateurs)**.**
Vous utiliserez la librairie `sqlite3` de Python pour manipuler les bases de données. Voir https://docs.python.org/3/library/sqlite3.html.
Votre script pourra par exemple débuter par des lignes comme celles-ci :

```
import sqlite3
#### Ouvrir la connection ou creer la base
connection = sqlite3.connect("donnees.db")
#### Creer un curseur
curseur = connection.cursor()
#### Creer la table : executer une seule fois
curseur.execute("CREATE TABLE utilisateurs (name TEXT, password TEXT)")
```

1. Écrivez une fonction `AjoutUtilisateur()` permettant d'entrer un login et un mot de passe en mode console. Elle doit vérifier que le login n'existe pas déjà dans la base, demander de taper deux fois le mot de passe, et stocker les deux valeurs en clair dans la table.
2. Après avoir exécuté au moins une fois `AjoutUtilisateur()`, affichez toutes les données stockées dans la base.
3. Écrivez une fonction `Verification()` permettant l'authentification d'un utilisateur en vérifiant si un login et un mot de passe entrés en mode console sont corrects.

Exercice 2.
En créant de nouvelles fonctions à partir des fonctions précédentes, modifiez votre stockage des mots de passe en les hachant avec SHA-256. Vous pourrez utiliser la bibliothèque `hashlib` de Python. Voir https://docs.python.org/3/library/hashlib.html. Attention au format des objets passés en paramètre et renvoyés par les commandes de cette librairie.

Exercice 3.
En créant de nouvelles fonctions à partir des fonctions précédentes, modifiez votre stockage des mots de passe en les hachant avec SHA-256 et en ajoutant un sel global que vous aurez choisi et écrit en dur dans votre code.

Exercice 4.
En créant de nouvelles fonctions à partir des fonctions précédentes, modifiez votre stockage des mots de passe en les hachant avec SHA-256 et en ajoutant un sel aléatoire que vous stokerez dans une autre

base de données. Vous pourrez utiliser la librairie `random` de Python pour générer un sel de 16 octets en hexadécimal (32 caractères hexa). Voir https://docs.python.org/3/library/random.html.

Exercice 5.
En créant de nouvelles fonctions à partir des fonctions précédentes, modifiez votre stockage pour utiliser pour le hachage la librairie `bcrypt` de Python. Voir https://github.com/pyca/bcrypt/blob/main/README.rst.

Exercice 6.
En créant de nouvelles fonctions à partir des fonctions précédentes, modifiez votre stockage pour chiffrer en plus les hachés des mots de passe avec une clef symétrique AES-256. Vous pourrez pour cela ajouter la ligne `from Crypto.Cipher import AES` au début de votre script (voir TP2 sur les chiffrements symétriques). Vous utiliserez le mode CTR et la clé `YELLOW SUBMARINE`. Voir https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html et https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#ctr-mode. Enfin, vous stockerez le `nonce` dans un champ supplémentaire de la table.

Exercice 7.
Dans cet exercice, vous allez évaluer les temps de vérification pour chacun des 6 types de stockage des mots passe et les comparer.

1. Écrivez une fonction `generermdp(N,n,alph)` permettant de générer `N` mots de passe aléatoires de `n` caractères pris dans l'alphabet `alph`. Vous pourrez pour cela utiliser la commande `choice` de la bibliothèque `random`. Vous stockerez les logins et les mots de passe en clair dans une table `utilisateurs`.
2. Pour chacun des 6 types de stockage des mots de passe vus dans les exercices précédents, écrivez une fonction permettant d'effectuer le stockage de l'ensemble du contenu de la table `utilisateurs` dans une nouvelle table.
3. Pour chacun des 6 types de stockage des mots de passe vus dans les exercices précédents, écrivez une fonction d'authentification d'un utilisateur sur la seule donnée de son login, en utilisant la table `utilisateurs` pour récupérer son mot de passe en clair.
4. Pour évaluer les différents temps de vérification, vous pourrez utiliser la bibliothèque `time` de Python. Voir https://docs.python.org/3/library/time.html. Vous évaluerez le temps d'authentification de l'ensemble des utilisateurs, pour une valeur de `N` assez élevée. À titre d'exemple, le mode de stockage le plus lent prend un temps de vérification de l'ordre de 10 secondes pour 50 utilisateurs sur ma machine.