Thibault GARROT

ICR - Laboratoire n°2

Gestionnaire de mots de passe

Ce laboratoire a pour objectif d'implémenter un gestionnaire de mots de passe sécurisé avec les bonnes pratiques cryptographiques.
Ce gestionnaire de mots de passe doit posséder les fonctionnalités suivantes :

- Afficher/Récupérer un mot de passe enregistré
- Changement du mot de passe maître
- Partage d'un mot de passe avec un autre utilisateur

En plus des fonctionnalités basiques :

- Création d'un compte
- Connexion à un compte

Le mot de passe maître est le seul et l'unique mot de passe que l'utilisateur doit connaître.

Un compte est identifié par un nom d'utilisateur ainsi qu'un mot de passe de maître afin de s'y connecter.
Le principal objectif est de sécuriser entièrement le mot de passe maître ainsi que les mots de passe enregistrés sur le gestionnaire.
Un compte possède deux états : l'état verrouillé et l'état déverrouillé

Etat verrouillé :

- Un attaquant ne doit pas être en mesure de retrouver les mots de passe, le mot de passe maître y compris.
- Un attaquant ne doit pas être en mesure de bruteforce le mot de passe maître non-trivial.

Etat déverrouillé :

- Un attaquant ne doit pas pouvoir retrouver le mot de passe maître dans la mémoire.
- Les mots de passe non demandé ne doivent pas être visible en clair.

Sommaire :

A. Modèle de sécurité

1. Gestion du mot de passe maître

Algorithmes utilisés :

- Argon2id (argon2-cffi 21.3.0) :
  - paramètres :
    - time(nombre d'itérations) = 1000
    - memory = 65536
    - parallelism = 4
    - hash_len = 32 bytes
    - salt_len = 16 bytes
- PRNG de la librairie Crypto.Random de Pycryptodome
- Chacha20-Poly1305, clé de 256 bits

La gestion du mot de passe maître est essentiel dans le gestionnaire de mots de passe, car en plus d'authentifier l'utilisateur sur son compte, il va nous permettre de générer les clés qui vont être utilisés pour chiffrer et déchiffrer les mots de passe que l'utilisateur va enregistré sur son gestionnaire.

Termes utilisés :

- Mot de passe maître : Mot de passe utilisé par l'utilisateur pour se connecter à son compte.
- Empreinte du mot de passe maître : Hash du mot de passe maître stocké en base de données.
- Clé maître : Deuxième hash du mot de passe maître utilisé comme clé de chiffrement. Non stocké en base de données, stockage de son sel.
- Clé des mots de passes : Clé utilisé pour chiffrer les mots de passe de l'utilisateur.

Création du compte

Lors de la création du compte, un nom d'utilisateur ainsi qu'un mot de passe maître va être demandé à l'utilisateur. Celui-ci va le taper deux fois pour être sûr qu'il n'ait pas fait d'erreur.

Ce mot de passe maître va être hashé grâce à l'algorithme Argon2id. Celui-ci génère un sel aléatoire pour chaque nouveau mot de passe ajouté. La version 2id maximise la résistance contre les attaques GPUs et les "side-channel attack". Le nombre d'itérations choisi pour Argon2 détermine le temps que l'on va mettre à hasher le mot de passe. Pour plus de sécurité, il est nécessaire d'augmenter ce nombre d'itérations. Le gestionnaire de mot de passe devant être un endroit sécurisé, j'ai décidé de faire 1000 itérations pour le mot de passe au détriment du temps mis à hasher (afin de tester l'application, il est possible de réduire ce nombre d'itérations pour que ce soit plus rapide). L'empreinte sera alors stocké avec le nom d'utilisateur dans la base de données.

Le mot de passe maître sera hashé une deuxième fois avec un sel différent pour générer la clé maître. Le sel sera stocké en base de données. La clé maître est ensuite utilisé pour chiffrer une clé de 256 bits générée aléatoirement grâce au PRNG de PyCryptodome avec Chacha20-Poly1305, utilisé pour chiffrer les futures mots de passe de l'utilisateur. Cette clé chiffrée est également stocké en base de données.

Une fois le processus terminé, la clé maître ne sera plus accessible.

Avantages de cette méthode :

- La fuite de l'empreinte du mot de passe maître (par dump de base de données) ne permet pas à un attaquant de déchiffrer les mots de passe de l'utilisateur.
- Le sel généré par Argon limite les bruteforces sur le mot de passe maître.
- Un changement du mot de passe maître n'impacte pas le chiffrement des anciens mots de passe. La clé utilisée pour les chiffrer ne changera pas, uniquement la clé maître sera modifié.

Améliorations possibles:

- Ajouter l'utilisation d'un poivre afin d'empêcher toutes tentatives de bruteforce.

Connexion au compte

A chaque fois que l'utilisateur se connecte, l'empreinte du mot de passe maître est vérifié. Si celle-ci est correcte alors l'utilisateur sera authentifié.
Nous récupérons la clé maître en effectuant le hash depuis le mot de passe maître et le sel stocké en base de données.
La clé de mots de passe est alors déchiffré et pourra être utilisé pour chiffrer ou déchiffrer les mots de passe de l'utilisateur.

Changement du mot de passe maître

Si l'utilisateur veut changer son mot de passe maître, il devra d'abord rentrer son ancien mot de passe. Quelques changements seront alors effectué :

- L'empreinte du mot de passe maître sera modifié en base de données.
- La clé maître sera modifié, donc un sel sera regénéré puis modifié en base de données.
- La clé des mots de passe sera déchiffré par l'ancienne clé maître puis rechiffré par la nouvelle.

2. Gestion des mots de passe

Algorithmes utilisés:

- Chacha20-Poly1305, clé de 256 bits
- RSA-OAEP, clé de 2048 bits

Comme expliqué précédemment, la clé de chiffrement des mots de passe est généré aléatoirement à la création du compte de l'utilisateur. Celle-ci est chiffré par la clé maître généré depuis le mot de passe maître de l'utilisateur. L'algorithme qui a été choisi pour le chiffrement de cette clé et de tout les mots de passe de l'utilisateur est Chacha20-Poly1305. Il nous permet l'authentification des mots de passe en plus du chiffrement. De plus, cet algorithme de chiffrement par flots est très simple d'utilisation et permet d'intégrer des données (header) supplémentaires dans le MAC. On verra dans notre cas que cette fonctionnalité est très utile. La taille de clé choisi est de 256 bits pour les deux processus de chiffrement (clé des mots de passe + mots de passe).

Enregistrer un nouveau mot de passe

Tout les mots de passe d'un utilisateur sont stocké dans un fichier qui lui est propre. Lorsqu'un utilisateur souhaite ajouter un nouveau mot de passe, nous allons utiliser Chacha20-Poly1305. Afin de garantir l'intégrité des données enregistrées, nous ajoutons le nom du site internet comme `header`. Cela nous permet de l'authentifier avec le mot de passe qui correspond, sans que ce dernier soit chiffré (car cela n'est pas nécessaire). De cette manière, un attaquant qui souhaiterait altérer les données en inversant les mots de passe des sites sera remarqué.

Améliorations possibles :

- Afin de protéger les mots de passe trop court, possibilité d'ajouter un padding avant de chiffrer. Ainsi tout les mots de passe on la même taille et il n'est pas possible de savoir combien de caractères il y a à bruteforcer.

Voir/Accéder aux mots de passe

Quand l'utilisateur est connecté, aucun mots de passe n'est stocké en clair. Ces derniers sont déchiffrés lorsque l'utilisateur en fait la demande. Il peut alors décider :

- d'afficher le mot de passe en clair
- de le copier dans le presse papier
- partager le mot de passe

Partager un mot de passe avec un utilisateur

La fonctionnalité partage de mot de passe est complexe à implémenter car elle nécessite de sécuriser l'échange entre deux utilisateurs. Afin de répondre à ce besoin, j'ai décidé d'utiliser l'algorithme de chiffrement asymétrique RSA-OAEP, appelé PKCS1_OAEP dans la librairie PyCryptodome.

- Lors de la création d'un compte utilisateur, une paire de clé RSA est créée est stocké dans des fichiers à part. Avant d'être stocké, la clé privé est d'abord chiffrée avec Chacha20-Poly1305 en utilisant la clé des mots de passe (celle généré grâce au mot de passe maître dérivé)
- Quand un utilisateur veut partager un mot de passe, il récupère la clé publique du destinataire. Le mot de passe qu'il souhaite partager est d'abord déchiffré avec sa propre clé avant de la rechiffrer avec la clé publique de RSA. Le module utilisé pour chiffrer avec RSA s'appelle PKCS1_OAEP.
- Quand le destinataire veut voir le mot de passe partagé, il récupère sa clé privé qu'il déchiffre avec sa propre clé, puis il déchiffre le mot de passe.
- Chaque utilisateur possède 2 fichiers pour les mots de passe partagés : un pour les mots de passe envoyé à quelqu'un, et un pour ceux reçus.

Cette implémentation ne protège pas la non-répudiation. Pour se faire, nous devons, en plus du chiffrement, signer le mot de passe envoyé. Cette signature peut se faire avec un algorithme de chiffrement asymétrique, typiquement RSA-OAEP ou ECC (Eliptic Curve Cryptography) de Pycryptodome.
L'utilisateur voulant partager son mot de passe vient signer le mot de passe chiffré avec sa clé privée. Le récepteur du mot de passe doit alors vérifier la signature avec la clé publique de l'envoyeur avant de déchiffrer le mot de passe. De cette manière, nous sommes sûr que celui qui partage le mot de passe est la bonne personne.
Par soucis de temps, cette fonctionnalité n'a pas été implémentée mais sera nécessaire afin de garantir une confiance total aux utilisateurs du gestionnaire de mots de passe.

B. Comment utiliser la gestionnaire de mots de passe

Application développée en Python

Version utilisée : Python 3.9.13

Librairies installées:

- PyCryptodome:
  - commande d'installation : "pip install pycryptodome"
- Argon2-cffi:
  - commande d'installation : "python -m pip install argon2-cffi"
  - documentation : https://argon2-cffi.readthedocs.io/en/stable/installation.html
- clipboard:
  - commande d'installation : "pip install clipboard"
- pandas:
  - commande d'installation : "pip install pandas" (version pip >=19.3)
- csv

Première utilisation

Pour créer tout les dossiers et les fichiers nécessaires au fonctionnement du gestionnaire, veuillez lancer la commande "python3 ./db_helper.py"

Lancer l'application

"python3 ./password_manager.py"