pwmgr

Gestionnaire de mots de passe local - Vault chiffre, CLI, presse-papiers auto-clear

Rapport technique et documentation utilisateur - 25/02/2026

Pitch 30 secondes

pwmgr est un gestionnaire de mots de passe minimal qui stocke vos comptes dans un **fichier coffre chiffre** (vault). L'utilisateur retient une seule phrase de passe (mot de passe maitre). Le programme permet d'ajouter, rechercher, copier un mot de passe dans le presse-papiers (avec effacement automatique), auditer la qualite des mots de passe, et faire des backups chiffres.

Public cible

Projet personnel oriente stage: montre une bonne maitrise des bases de la securite (chiffrement, KDF), une architecture simple, une CLI propre, des tests et une documentation.

Sommaire

1. Contexte et objectifs

Le besoin: eviter les oublis et la reutilisation de mots de passe, tout en gardant le controle sur ses donnees. Plutot qu'une plateforme complexe, pwmgr est concu comme un outil local, simple a utiliser depuis le terminal.

Objectifs du MVP

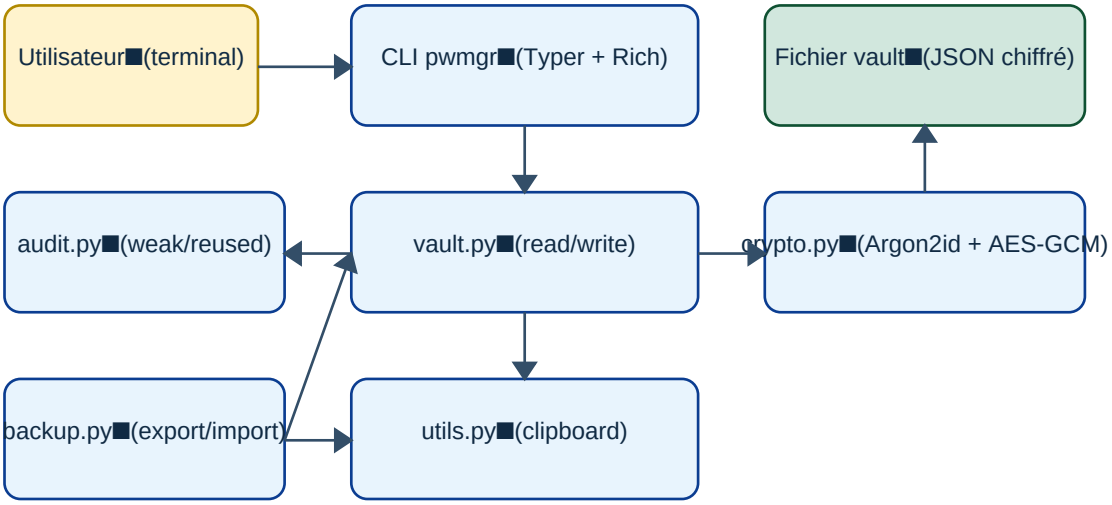
- Un seul fichier vault chiffre (pas de mots de passe en clair sur disque).
- Commandes basiques: init, add, list, show, copy, update, delete.
- Workflow rapide: copier le mot de passe puis coller sur un site.
- Audit (faibles / reutilises) et backup (export/import).

2. Fonctionnalites principales

Fonction	Interet
Vault chiffre local	Toutes les entrees sont stockees dans un fichier JSON contenant uniquement du texte chi
Ajout et recherche	Ajout d'un compte (title, username, password, url, notes, tags). Recherche via 'list --q'.
Copie au presse-papiers	Commande 'copy' qui copie le mot de passe et l'efface automatiquement apres N seconde
Audit	Detection de mots de passe trop courts / peu varies et detection de reutilisation.
Backup/restore	Export d'un backup chiffre et import dans un nouveau vault. Option de re-chiffrement avec
Change-master	Changement du mot de passe maitre via re-chiffrement complet du vault.

3. Architecture (vue d'ensemble)

L'architecture est volontairement simple: une CLI qui charge/dechiffre le vault, applique l'action demandee, puis rechiffre et sauvegarde. Les modules sont separes pour clarifier les responsabilites.



Separation des responsabilites:
• **cli.py** orchestre les commandes et l'IHM terminal.
• **vault.py** gere lecture/ecriture du fichier et la sauvegarde atomique.
• **crypto.py** encapsule KDF + chiffrement/dechiffrement.
• **utils.py** gere generation de mots de passe + presse-papiers.
• **audit.py** analyse la qualite des mots de passe (sans les afficher).
• **backup.py** export/import et re-chiffrement.

4. Modele de donnees

Chaque entree est un objet 'Entry'. Le vault contient un dictionnaire {id -> Entry}.

Champs d'une entree

- **title**: nom lisible (ex: GitHub).
- **username**: identifiant / email.
- **password**: mot de passe (en memoire uniquement; chiffre sur disque).
- **url** (optionnel), **notes** (optionnel), **tags** (liste).
- **created_at**, **updated_at**: timestamps UTC ISO8601.

5. Format du vault (fichier coffre)

Le vault est un fichier JSON qui ne contient pas les donnees en clair. Il contient un en-tete (parametres KDF + salt) et un bloc 'ciphertext'.

vault.json (JSON)

```
"version": 1,
"kdf": { type: "argon2id", time_cost, memory_cost_kib, parallelism, ... },
"salt": "<base64>",
"nonce": "<base64>",
"ciphertext": "<base64>"   // JSON des entrees, chiffre
```

Important:

Aucun mot de passe en clair dans ce fichier.

Pourquoi du JSON chiffre ?

Pour un MVP, chiffrer un seul blob JSON est simple, fiable, et facile a sauvegarder. Une evolution possible est un stockage SQLite chiffre ou un format de vault plus complexe, mais ce n'est pas necessaire pour demontrer les bonnes pratiques.

6. Cryptographie et securite

pwmgr n'invente pas de cryptographie: il s'appuie sur des bibliotheques reconnues. Deux briques principales: une fonction de derivation de cle (KDF) et un chiffrement authentifie.

6.1 Derivation de cle (Argon2id)

Le mot de passe maitre n'est pas une cle cryptographique. On le transforme en cle via Argon2id. Argon2id est concu pour etre couteux en memoire et en temps, ce qui ralentit fortement les attaques par force brute (GPU/ASIC).

Parametres utilises (exemple)

- time_cost=3 (nombre de passes).
- memory_cost_kib=65536 (64 MiB).
- parallelism=2.
- salt=16 octets aleatoires, stocke dans le vault.

6.2 Chiffrement authentifie (AES-256-GCM)

Une fois la cle derivee, les donnees (JSON des entrees) sont chiffrees avec AES-GCM. AES-GCM fournit deux proprietes: (1) confidentialite et (2) integrite/authenticite. Si le fichier est modifie, le dechiffrement echoue.

AAD (Authenticated Additional Data)

Le header (version, kdf params, salt) est ajoute comme donnees authentifiees. Cela lie le header au ciphertext et evite des manipulations subtiles (ex: changer des parametres).

6.3 Menaces couvertes et limites

Couvert

- Vol du fichier vault: sans le mot de passe maitre, le contenu reste inutilisable.
- Corruption / modification du fichier: detectee par AES-GCM (InvalidTag).

Non couvert (a expliquer a un recruteur)

- Malware/keylogger: si le poste est compromis, le mot de passe maitre peut etre vole.
- Memoire: pendant l'execution, les secrets sont en RAM (comme tous les password managers).
- Presse-papiers: d'autres applis peuvent lire le contenu copie (risque connu).

7. Bibliotheques utilisees

Bibliotheque	Role
Typer	Framework CLI (base sur Click). Permet des commandes lisibles et des options propres.
Rich	Affichage terminal colore: tableaux, messages plus agreables.
cryptography	Bibliotheque de reference pour AES-GCM et primitives securisees.

argon2-cffi	Implementation Argon2id pour la derivation de cle.
pyperclip	Acces portable au presse-papiers (copy/paste) + auto-clear.
pytest	Framework de tests unitaires.

8. Reference des commandes CLI

Toutes les commandes acceptent --path pour choisir le fichier vault.

Commande	Utilite	Exemple
init	Cree un vault vide chiffre	python -m pwmgr init --path .\.vault.json
add	Ajoute un compte (option --gen)	python -m pwmgr add --path .\.vault.json --title GitHub --usernar
list	Liste ou recherche	python -m pwmgr list --path .\.vault.json --q github
show	Affiche une entree (mot de passe masque)	python -m pwmgr show GitHub --path .\.vault.json
copy	Copie le mot de passe + auto-clear	python -m pwmgr copy GitHub --path .\.vault.json --seconds 20
update	Modifie une entree	python -m pwmgr update GitHub --path .\.vault.json --ask-passw
delete	Supprime une entree	python -m pwmgr delete GitHub --path .\.vault.json
audit	Mdp faibles / reutilises	python -m pwmgr audit --path .\.vault.json
export	Backup chiffre	python -m pwmgr export --path .\.vault.json --out .\backup.vault.
import / import_	Restaure un backup (nom exact via --help)	python -m pwmgr import --in .\backup.vault.json --path .\.vault.js
change-master	Change le mot de passe maitre	python -m pwmgr change-master --path .\.vault.json

Astuce demo (workflow en 15 secondes)

1) python -m pwmgr copy GitHub --seconds 20
2) Colle sur le site (Ctrl+V).
3) Le presse-papiers est nettoye automatiquement si tu ne copies rien d'autre.

9. Tests et qualite

Des tests unitaires simples valident la robustesse du coeur securite: roundtrip chiffrement/dechiffrement et roundtrip ecriture/lecture du vault.

```
pytest -q
# test_crypto.py: encrypt -> decrypt, mauvais mot de passe -> erreur
# test_vault_roundtrip.py: init vault -> add entry -> save -> load -> verif
```

10. Decisions de conception (choix)

Pourquoi une CLI (pas une GUI) ?

Une CLI est rapide a developper et parfaite pour un MVP. Elle permet aussi une demo simple. Une GUI peut etre ajoutee ensuite (Tkinter, Electron/Tauri) sans changer le coeur (vault/crypto).

Pourquoi un vault monolithique (un blob chiffre) ?

Simplicite et fiabilite: une seule operation de chiffrement et une seule ecriture atomique. Moins de risques de bugs sur un projet stage. Evolution possible vers un stockage indexe.

11. Roadmap (ameliorations possibles)

- Verrouillage automatique (session) et timeout d'inactivite.
- Integration OS/browseur (extension Chromium ou service autofill).
- Historique de mots de passe (avec chiffrement) et rotation.
- Export CSV (attention: non chiffre) uniquement sur demande et avertissement clair.
- Support multi-vault (perso/pro) via config.

12. FAQ recruteur - questions frequentes

Pourquoi Argon2id ?

Parce que c'est une KDF moderne concue pour etre couteuse en memoire, donc plus resistante aux attaques GPU que PBKDF2.

Comment detectes-tu la corruption du vault ?

AES-GCM fournit un tag d'integrite. Si le fichier est modifie, le dechiffrement echoue (InvalidTag).

Pourquoi le presse-papiers est un risque ?

Certaines applications peuvent lire le presse-papiers. On limite le risque via auto-clear apres N secondes et on n'efface pas si l'utilisateur a copie autre chose entre temps.

Que se passe-t-il si je perds le mot de passe maitre ?

Les donnees sont chiffrees. Sans le mot de passe maitre, on ne peut pas recuperer le contenu. D'ou l'importance des backups et d'une phrase de passe memorisable.

Comment changes-tu le mot de passe maitre ?

On dechiffre le vault avec l'ancien mot de passe, puis on rechiffre l'ensemble avec le nouveau (change-master).

13. Mettre le projet sur GitHub (pas a pas)

Ces commandes sont valables sous Windows/PowerShell.

```
# 1) Initialiser Git
git init
git add .
git commit -m "Initial commit: pwmgr (vault chiffre + CLI)"

# 2) Creer un repo sur GitHub (UI) puis recuperer l'URL, ex:
# https://github.com/<user>/pwmgr.git

# 3) Ajouter le remote et pousser
git branch -M main
git remote add origin https://github.com/<user>/pwmgr.git
git push -u origin main
```

Conseil

Ajoute le vault au .gitignore (ex: .vault.json) pour ne jamais pousser des secrets (meme chiffres) par erreur. Garde un README clair et un historique de commits propre.