Password Manager — Plan d'action complet (PC & Mobile)

> Objectif global : transformer ton gestionnaire de mots de passe en application distribuable pour PC (Windows/Linux/macOS) puis mobile (Android/iOS), avec une base de code pédagogique, testée et sécurisée.

Méthode de travail (pédago)

Chaque étape suivra toujours ce format : 1) **Objectif** → ce qu'on veut obtenir. 2) **Tu vas coder** → le périmètre exact de ce que *toi* tu écris. 3) **Pense-bête** (mini-cours) → rappels ultra concrets (classe, fonction, exceptions, tests…). 4) **Checklist** → cases à cocher avant de passer à l'étape suivante. 5) **Validation** → comment vérifier que ça marche (scripts, tests, manip GUI…).

Phase 0 — Audit & stabilisation de la base (Desktop Tkinter)

Objectif : figer le socle, clarifier le flux "mot de passe maître → dérive clé → chiffrer/déchiffrer vault JSON".

Tu vas coder : - Structurer le projet (src/ tests/ assets/), `main.py` point d'entrée, `gui.py` interface, `utils.py` crypto/IO. - Introduire une classe `Vault` (logique cœur sans GUI) + gestion d'erreurs propre. - Ajouter logs (niveau DEBUG/INFO) & type hints.

Pense-bête : - *Classe* : état + méthodes ; **init** pour initialiser. - *Fonction* : fait une seule chose ; arguments typés ; docstring courte. - *Exceptions* : `try/except` pour protéger l'I/O & crypto.

Checklist ✅ - [ ] Structure de dossiers propre (src/gui.py, src/utils.py, src/vault.py, src/main.py, tests/…) - [ ] Fichier `requirements.txt` - [ ] `Vault` : charger/sauver un fichier **entièrement chiffré** (pas de JSON en clair sur disque) - [ ] Journalisation minimale (`logging`) - [ ] Fenêtre Tkinter redimensionnable (astuce Treeview)

Validation : lancer l'app, créer 1 entrée, quitter, relancer, déchiffrer et retrouver l'entrée.

Phase 1 — Sécurité & chiffrement

Objectif : rendre la crypto robuste et claire.

Tu vas coder : - Dérivation de clé (PBKDF2HMAC existante) avec paramètres constants et `salt` stocké à part. - Chiffrement symétrique (Fernet) d'un bloc unique « vault.json.enc ». - Effacement du presse-papiers après X sec, masquage/révélation mot de passe.

Pense-bête : - *PBKDF2* : transforme le mot de passe maître en clé ; plus d'itérations = plus coûteux pour un attaquant. - *Salt* : public, random, empêche les rainbow tables. - *Fernet* : inclut HMAC → intégrité + timestamp.

Checklist ✅ - [ ] `derive_key(master_password) -> bytes` - [ ] `Vault.load(path, key)` / `Vault.save(path, key)` - [ ] Fichier chiffré unique + salt séparé - [ ] Clipboard auto-clear

Validation : tests unitaires sur dérivation & round-trip (chiffre→déchiffre).

Phase 2 — Fonctionnalités de base (CRUD + UX)

Objectif : gérer les entrées de manière fluide.

Tu vas coder : - CRUD complet : ajouter/éditer/supprimer. - Recherche instantanée, tri par colonnes. - Générateur de mot de passe + indicateur de robustesse. - Tags/catégories.

Pense-bête : - *Séparation des responsabilités* : GUI déclenche, `Vault` exécute et valide.

Checklist ✅ - [ ] Modale « nouvelle entrée » stable (validations) - [ ] Recherche filtrante - [ ] Générateur + barre de force - [ ] Tags (liste simple)

Validation : tests sur `Vault` (ajout/édition/suppression), tests manuels GUI.

Phase 3 — Qualité : tests, lint, typage, logs

Objectif : code fiable et maintenable.

Tu vas coder : - `pytest` + `hypothesis` (fuzz léger sur générateur mdp). - `ruff` + `black` + `mypy`. - Journalisation d'erreurs dans un fichier rotatif.

Checklist ✅ - [ ] `pytest -q` passe - [ ] `ruff --fix` et `black` OK - [ ] `mypy` sans erreurs

Validation : CI locale (script make ou taskfile).

Phase 4 — Polissage UX (Desktop)

Objectif : app agréable et claire.

Tu vas coder : - Thème ttk, icônes, raccourcis clavier (Ctrl+N, Ctrl+F, Del…). - Dialogues d'alerte/confirmation propres. - Écran de verrouillage rapide (temporisation).

Checklist ✅ - [ ] Thème uniformisé - [ ] Raccourcis fonctionnels - [ ] Verrouillage automatique après N minutes

Validation : revue UX rapide + tests manuels.

Phase 5 — Emballage Desktop (distribution)

Objectif : produire des exécutables installables.

Tu vas coder : - Windows : PyInstaller (onefile + dossier assets), icône, UAC-friendly. - Linux : AppImage ou .deb ; dépendances. - macOS : .app via PyInstaller ; notarisation (optionnel).

Checklist ✅- [ ] Script `build_win.sh` / `build_linux.sh` / `build_mac.sh` - [ ] Inclus : icônes, fichier conf, licence - [ ] Test sur VM/minimum runtime

Validation : lancer l'exécutable sur machine « vierge ».

Phase 6 — Synchronisation & sauvegardes (optionnel)

Objectif : utiliser l'app sur plusieurs machines.

Tu vas coder : - Export/Import chiffré. - Option "dossier de synchronisation" (ex. Nextcloud/Dropbox/WebDAV) *sans* envoyer la clé. - Sauvegardes chiffrées datées.

Checklist ✅- [ ] Export/Import OK - [ ] Choix répertoire sync - [ ] Rotation des sauvegardes

Validation : restaurer un coffre sur une autre machine.

Phase 7 — Mobile : choix de la voie

Objectif : porter sur Android/iOS.

Deux options réalistes : - **A. Kivy** (Android très mature, iOS possible) : refaire la GUI en Kivy, garder `Vault` commun. - **B. BeeWare / Toga** (Android/iOS/desktop natifs) : GUI Toga, même logique partagée.

Recommandation : conserver Tkinter pour Desktop, extraire **toute la logique** dans un module `vault/` (sans dépendance GUI), puis créer une **seconde interface** (Kivy ou Toga) qui réutilise ce module.

Checklist ✅- [ ] `vault/` packagée (setup.cfg/pyproject) - [ ] GUI Desktop = Tkinter, GUI Mobile = Kivy/Toga - [ ] Build Android avec Buildozer (Kivy) **ou** Briefcase (BeeWare)

Validation : APK installée, déverrouillage, CRUD d'une entrée.

Phase 8 — Mises à jour & signature (plus tard)

Objectif : expérience de distribution propre.

- Signature de l'exécutable (Windows, macOS).

• Mise à jour semi-auto (vérifier version + proposer téléchargement).
• Crash reports (fichiers logs anonymisés).

Phase 9 — Documentation & fiches pense-bête

Objectif : t'outiller pour apprendre en faisant.

À livrer au fil de l'eau : - **Fiche "Classe"** : attributs, méthodes, **init**, encapsulation, propriétés. - **Fiche "Fonction"** : signature, args nommés, valeur de retour, docstring, exceptions. - **Fiche "Exceptions"** : try/except, raise, types d'erreurs, messages utiles. - **Fiche "Tests"** : unitaires, paramétrés, fixtures, property-based (Hypothesis). - **Fiche "Crypto"** : PBKDF2, salt, Fernet, bonnes pratiques (jamais stocker la clé en clair, effacer le presse-papiers, etc.). - **Fiche "Packaging"** : PyInstaller/AppImage/Briefcase, icônes, ressources.

Prochaine étape (concrète)

1) **On extrait la logique cœur** dans `vault/` : - `vault/__init__.py`, `vault/model.py` (classe `Vault`), `vault/crypto.py`, `vault/io.py`. 2) **GUI Tkinter** appelle uniquement `vault` (plus d'accès direct au JSON/crypto depuis la GUI). 3) **Test unitaire minimal** : round-trip chiffré (créer coffre → sauvegarder → recharger → égalité des données).

Quand tu veux, on démarre par *Phase 0* et je te guide en pas-à-pas 💟