

CAA 2022 Labo 2 - Password manager

David Pellissier

Remarques

Avant de commencer à utiliser le password manager, s'assurer que le dossier `./users` existe bien à la racine du projet. Si besoin, le chemin peut être configuré dans le module `utils::config`.

L'output fonctionne mieux sur un terminal que sur la console JetBrains (mais ça reste utilisable).

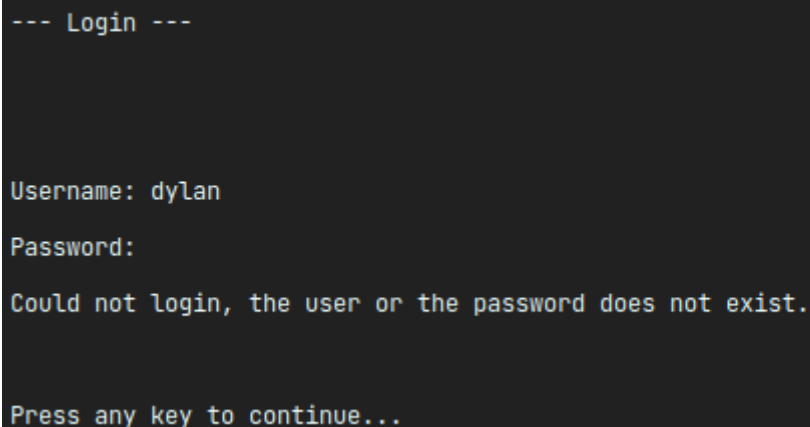
Fonctionnement de base

Fonctionnalités

Utilisateur non connecté

- **Login**

Le login est très simple. Le programme demande un nom d'utilisateur et un mot de passe (non affiché à l'écran) et renvoie un message d'erreur générique s'il y a des mauvais credentials.



```
--- Login ---

Username: dylan
Password:
Could not login, the user or the password does not exist.

Press any key to continue...
```

Lorsque l'utilisateur entre ses credentials, deux clés sont dérivées:

1. La master key, qui sert à vérifier l'intégrité et déchiffrer la base de données. Elle n'est pas gardée en mémoire (voir "Autres mesures implémentées").
2. La passwords key, qui sert à déchiffrer la partie chiffrée des mots de passes. Elle est stockée en mémoire pour la session.

- **Créer un utilisateur**

Pour que l'utilisateur soit créé, il faut qu'il n'existe pas encore, et que les deux mots de passes entrés soient égaux et satisfassent la politique de mots de passe.

La politique de mots de passe peut être modifiée dans le module de configuration. Pour faciliter les tests, j'ai mis une politique très simple de 8 caractères, qu'il vaudrait mieux éviter en pratique.

```
--- Ernest ---

login : Log in
add   : Add a new user
exit  : Exit

What do you want to do ? a
Name of the new user: charlie
Enter your password:
Confirm your password:
These two passwords don't match. Please try again.
Enter your password:
Confirm your password:
This password does not match the password policy (defined in utils.rs)
Enter your password:
Confirm your password:

Creating your user ...
User added ! (./users/charlie)

Press any key to continue...
```

Utilisateur connecté:

- **Lister les mots de passe enregistrés**

Les mots de passes ne sont pas affichés en clair (uniquement leurs autres attributs).

L'utilisateur doit en sélectionner un parmi la liste qui va ensuite être déchiffré puis affiché.

```

--- Your passwords ---

N°, Label || Username
-----
0 : test || user
1 : gmail || dav.pellissier@gmail
2 : facebook || pablob

Select a password (>=3 to cancel) : 2
Password with label 'facebook' and username 'pablob' is :    superman

Press any key to continue...

```

- **Ajouter ou mettre à jour un mot de passe**

Si la combinaison mot de passe + utilisateur ajouté existe déjà, alors le programme demande s'il doit être mis à jour.

Il n'y a pas de politique de mots de passes car je pense que cela pourrait être contre l'intention de l'utilisateur et la plupart des sites possèdent eux-mêmes une politique de mots de passe.

```

--- Add a new password ---

Label: test
Username (optional): user
Password: lalala
This label/username combination already exists.
Do you want to update the password ? [N/y]y
Password added to the database

Press any key to continue...

```

- **Supprimer un mot de passe**

L'utilisateur doit confirmer l'action avant que la suppression soit faite.

- **Partager un mot de passe**

Les mots de passes partagés sont chiffrés avec la clé publique du destinataire et ajoutés dans son fichier partagé `shared.nest`.

```

--- Password sharing ---

Who do you want to share the password to ? dylan
Your passwords:
N°, Label || Username
-----
0 : gmail || dav.pellissier@gmail
1 : facebook || pablob
2 : test || user

Which password do you want to share ? (0-2) 0
Selected user: dylan
Selected password: gmail with username 'dav.pellissier@gmail'
Do you want to continue ? [Y/n]
Password shared successfully.

Press any key to continue...

```

- **Changer de master password**

Tous les mots de passes sont re-chiffrés et le fichier de stockage est réécrit. L'utilisateur est automatiquement déconnecté.

- **Importer les mots de passes partagés**

Cela se fait automatiquement lors du login de l'utilisateur. Si des mots de passe ont été partagés, le programme demande s'il doit les importer.

```

You have 2 shared passwords waiting to be imported.

Do you want to import them ? [Y/n]y
Imported successfully.

```

- **Sauvegarder les modifications**

Les modifs apportées à la base de données ne sont pas écrites automatiquement afin de ne pas avoir à stocker la master key en mémoire. Cela implique que l'utilisateur doit à nouveau rentrer son master password pour pouvoir re-chiffrer et écrire les données dans son "nest file".

Un message est ajouté lorsque la base de données a été modifiée afin de rappeler cela.

```
Connected since 2022-05-29T22:52:56.-31600
[!] The database was modified. It will be saved at logout or manually with 'write'

--- Menu ---

get    : Get passwords
add    : Add a password
remove : Remove a password
share  : Share a password with another user

change : Change your master password
write  : Apply changes of the nest file

exit   : Log out and exit

What do you want to do ?
```

- **Déconnexion**

S'il y a eu des modifications, le programme demande à l'utilisateur s'il veut enregistrer sa base de données. Comme pour la sauvegarde manuelle, l'utilisateur doit entrer son master password.

```

--- Menu ---

get      : Get passwords
add      : Add a password
remove   : Remove a password
share    : Share a password with another user

change   : Change your master password
write    : Apply changes of the nest file

exit     : Log out and exit

What do you want to do ? exit

Your nest file has changed. Do you want to save it ? [Y/n]

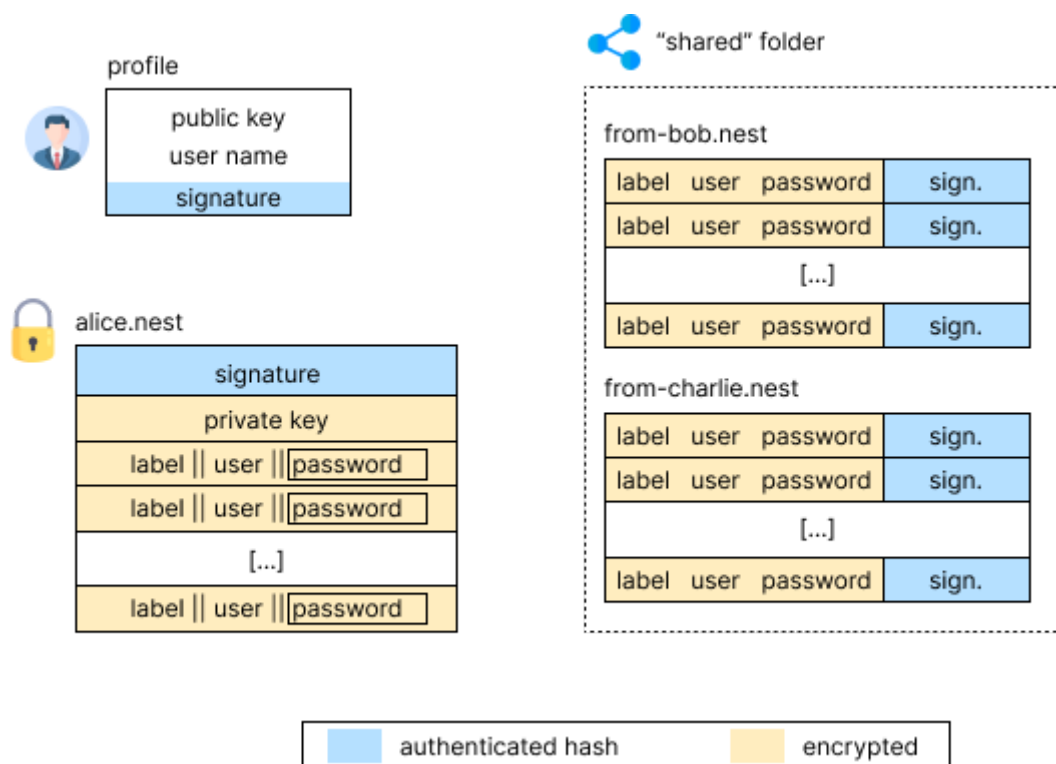
Please enter your master password or keep empty to cancel :

Nest saved successfully !

Press any key to continue...

```

Structure des fichiers



REMARQUE: CE SCHÉMA VIENT DU PROTOTYPE, le dossier shared est devenu un fichier. Je n'ai pas eu le temps de mettre à jour le schéma mais le principe reste quasiment le même.

Les fichiers sont sérialisés avec MessagePack, qui a l'avantage d'être très léger : <https://msgpack.org/>

Lors de la création d'un compte, trois fichiers sont créés au sein d'un répertoire ayant le nom de l'utilisateur-

1. profile

Le fichier `profile` contient les données du profil public de l'utilisateur qui sont utiles au partage de mot de passe:

- Nom d'utilisateur
- Clé publique
- Signature du nom et de la clé publique

2. {user}.nest

Le fichier `<user>.nest` est le fichier privé de stockage des mots de passes de l'utilisateur. Il est chiffré et authentifié grâce à master password.

On y retrouve les données suivantes:

- Clé privée pour le partage de mots de passes
- Liste des mots de passes. Les mots de passes sont des objets qualifiés par les attributs suivants:
 - un label (exemple: `gmail`)
 - le nom d'utilisateur (exemple: `alice@gmail.com`)
 - le mot de passe chiffré

3. shared.nest

Le fichier partagé contient la liste tous les mots de passes partagés à l'utilisateur.

Chaque mdp partagé est constitué des valeurs suivantes:

- Chiffré avec la clé publique de Alice:
 - label
 - user
 - password
- Non chiffré:
 - Nom de l'émetteur, afin que le programme puisse aller chercher la clé publique dans son profil
 - Signature du contenu chiffré, signé par l'émetteur

Lorsque l'utilisateur se connecte, l'application lui demande s'il souhaite importer les mots de passes, et si oui, il va alors vérifier puis déchiffrer avec la clé privée les mots de passes. Le fichier est vidé une fois l'importation faite.

Structure du code

Le code est structuré en plusieurs modules:

- `passwords` : Tout ce qui concerne la gestion des mots de passes enregistrés.
 - struct `EncryptedPassword`
 - struct `UnencryptedPassword`
 - module `crypto` : dé/chiffrement d'un password avec une clé symétrique

- `session` : Concerne la session en active de l'utilisateur
 - struct `Session` : avec tous les handlers qui permettent à l'utilisateur d'interagir avec les fonctionnalités
 - struct `DecryptedData` : ce qui est stocké en mémoire lorsque la session est active
 - struct `EncryptedData` : ce qui est écrit dans un nest file
 - module `crypto` : pour la dérivation de clé, et le chiffrement de la base de données
- `share_passwords` : fonctionnalité de partage de mots de passes
 - struct `SharedPassword`
 - fonctions de récupération et d'importation de mots de passes partagés
 - module `crypto` : pour le chiffrement asymétrique
- `users` : fonctionnalités applicables aux utilisateurs
 - struct `PublicProfile`
 - struct `UserFiles` : décrit les différents fichiers de l'utilisateur
 - création de compte
 - module `crypto` : génération de la pair de clés asymétrique
- `utils` : fonctionnalités diverses utilisées un peu partout.
 - Les fonctions sont triées par ordre alphabétique
 - module `config` : définit la configuration de certains paramètres

Sécurité

Choix des algorithmes

Utilisation	Algo	Tailles	Remarque	Crate
Dérivation de clés	PBKDF2	-	+ Sel	<code>pbkdf2</code> v0.11
Chiffrement BDD	Chacha20-Poly1305	nonce: 12 bytes	-	<code>chacha20poly1305</code> v0.10.0-pre
Chiffrement mdp	AES-GCM avec AES-256	nonce: 12 bytes	-	<code>aes-gcm</code> v0.9.4
Signatures mdp partagés	RSA-PSS avec SHA256	-	-	<code>rsa</code> v0.6.1 et <code>sha2</code> v0.10.2
Chiffrement mdp partagés	RSA-OAEP avec SHA256	-	-	<code>rsa</code> v0.6.1 et <code>sha2</code> v0.10.2

- S'il y a des infos manquantes, c'est que j'ai utilisé les paramètres par défauts des crates.

Autres mesures implémentées

Politique de Master Password

Une politique de mots de passes est appliquée sur le master password.

Elle est configurable dans le module `config` se trouvant à la fin du fichier `utils.rs`

```
pub mod config {  
    // DO NOT CHANGE THIS after having created a user  
    pub const PBKDF_SALT: &str = "isntthatpepper";  
    // simple regex for testing purpose. Please note that only the master password is checked with this regex.  
    pub const PWD_REGEX: &str = r"^[0-9a-zA-Z]{8,64}$";  
    pub const USERS_PATH: &str = "./users";  
}
```

Mots de passes chiffrés en mémoire

Au sein de la base de données chiffrée, les mots de passes sont re-chiffrés avec AES-GCM puis déchiffrés uniquement au moment de l'affichage à l'utilisateur. Cela permet de ne pas leak de mot de passe en cas de buffer overflow.

Pour le déchiffrement, j'utilise la `passwords_key` qui est dérivée du Master Password concaténé à une constante. La passwords key est quand même stockée en mémoire car il n'y a pas d'autre solution (à ma connaissance) si on ne veut pas demander le Master Password à chaque fois qu'on affiche un mdp ou alors utiliser un système de mémoire protégée.

Stockage des clés

Je ne stocke pas la Master Key, mais je dérive une Passwords Key qui sert à déchiffrer les mots de passes stockés en mémoire.

Cela implique qu'en cas de dump mémoire, les mots de passes pourront être déchiffrés par un attaquant via la Passwords Key, mais vu que la master_key n'a pas leak, seuls les passwords capturés au moment du dump seront compromis. Les mots de passes modifiés et rajoutés par la suite ne pourront pas être déchiffrés sans un autre dump mémoire.

Input de master password caché

Grâce à la crate `rpassword`, les inputs du mot de passe de session sont cachés.

```
--- Ernest ---

login : Log in
add   : Add a new user
exit  : Exit

What do you want to do ? a
Name of the new user: pablo
Enter your password:
Confirm your password:

Creating your user ...
User added ! (./users/pablo)

Press any key to continue...
```

Impact du langage Rust

Rust a plusieurs impacts au niveau de la sécurité du programme:

1. Les lifetimes, qui permettent de mieux contrôler l'utilisation de la mémoire et la durée de vie des données.
2. Typage de variables strict, qui protège plus des bugs de conversion de type et de "integer overflow"
3. Le compilateur trouve beaucoup plus facilement des bugs bas niveau, ce qui fait que certains bugs d'exécution sont immédiatement détectés avant même d'avoir pu compiler

Pour aller plus loin

1. Avoir un temps constant pour toutes les actions sensibles
 - Afin d'éviter les side channel attack et timing attack, il faudrait que le code soit constant en temps autant que possible. La crate [subtle](#) permet cela.
2. Verrouiller automatiquement la session après un certain temps
 - Il est dangereux mais malheureusement fréquent pour un utilisateur de laisser sa session ouverte et quitter sa place de travail. Quelqu'un de mal intentionné pourrait alors se servir de la session pour récupérer tous les mots de passes. Afin d'éviter cela, il serait judicieux de fermer la session de manière automatique lorsqu'il n'y a plus eu d'actions après un certain temps. Vu que mon implémentation demande à rentrer le master password pour ne pas perdre de données à la déconnexion, on pourrait simplement verrouiller la session sans la fermer, et demander à l'utilisateur se réauthentifier ou abandonner les changements.
 - L'attribut `login_time` de la struct `Session` a été créé dans cet objectif.
3. Copier les mots de passe dans le presse-papier au lieu de les afficher:

- C'est une sécurité d'un point de vue visuel car personnes externes ne pourront pas voir la valeur du mot de passe dans le terminal.
4. Vider le presse-papier automatiquement:
- Lors du verrouillage du compte ou après un timer, le presse-papier pourrait être vidé afin d'empêcher que les mots de passes déchiffrés ne puissent leak après que l'application se soit arrêtée
5. Gestion des permissions des fichiers:
- Les permissions des fichiers pourraient être définies lors de la création de manière à ne pas pouvoir être lus par les utilisateurs non propriétaires.

J'ai essayé d'implémenter les points 3 et 4 en m'inspirant de [ce code](#), mais je suis tombé sur un problème de dépendances Linux qui m'empêchait de compiler:

```
error: linking with `cc` failed: exit status: 1
note: "cc" "-m64" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.101psodb1dh64ood.rcgu.o" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.107k541bj1lpzsnr.rcgu.o" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.101f6c4c077qastu.rcgu.o" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.11126duhky315.rcgu.o" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.11djmhco7xpx3nu.rcgu.o" "/home/david/Cours/BA_ALL/BA6_CAA/labo/12_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00.11djmhco7xpx3nu.rcgu.o" "-Wl,--gc-sections" "-pie" "-Wl,-zrelro,-znow" "-nodefaultlibs"
```

Fin du message:

```
l2_password_manager/ernest/target/debug/deps/ernest_passwords-6325fb21d4590c00 "-Wl,--gc-sections" "-pie" "-Wl,-zrelro,-znow" "-nodefaultlibs"
note: /usr/bin/ld: cannot find -lxcb-render: No such file or directory
/usr/bin/ld: cannot find -lxcb-shape: No such file or directory
/usr/bin/ld: cannot find -lxcb-xfixes: No such file or directory
collect2: error: ld returned 1 exit status
warning: 'ernest_passwords' (bin 'ernest_passwords') generated 22 warnings
error: could not compile 'ernest_passwords' due to previous error; 22 warnings emitted
~/Cours/BA6_CAA/labo/12_password_manager/ernest(main)?@budp2204
```

J'ai donc installé les dépendances `xorg-dev` et `libxcb1-dev`:

```
> apt search xorg-dev
2204
Sorting... Done
Full Text Search... Done
xorg-dev/jammy,jammy,now 1:7.7+23ubuntu2 all [installed]
X.Org X Window System development libraries

xserver-xorg-dev/jammy,now 2:21.1.3-2ubuntu2 amd64 [installed,automatic]
Xorg X server - development files

xserver-xorg-dev-hwe-18.04/jammy 3:14.6 amd64
Transitional package for xserver-xorg-dev-hwe-18.04

> apt search libxcb1-dev
2204
Sorting... Done
Full Text Search... Done
libxcb1-dev/jammy,now 1.14-3ubuntu3 amd64 [installed,automatic]
X C Binding, development files

> _
```

Mais ça n'a pas suffi à résoudre mon problème. J'ai donc abandonné cette fonctionnalité.