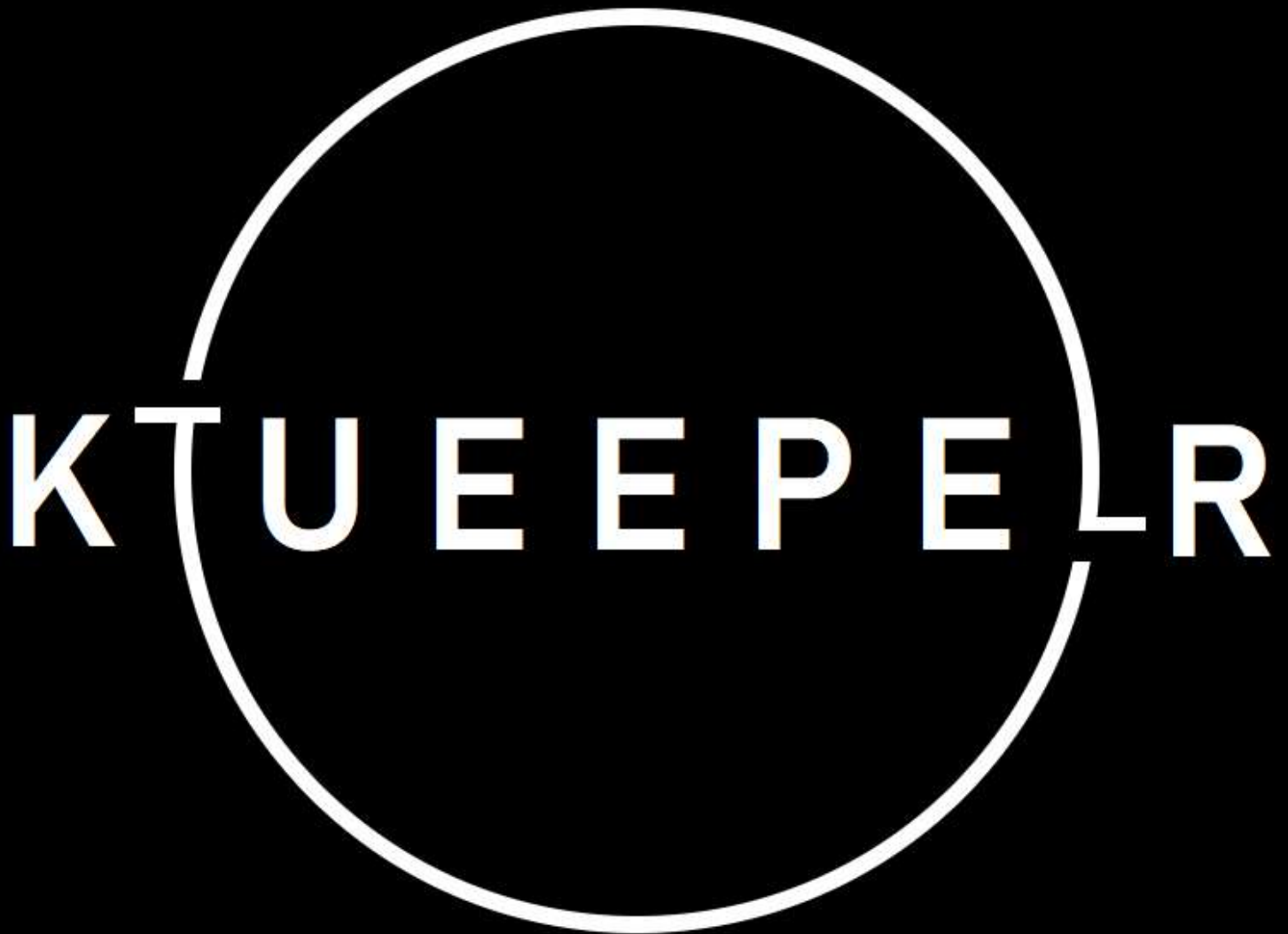


Rapport de fin de Projet:
Introduction à la sécurité:
Gestionnaire de mot de passe



Prérequis :

Le gestionnaire, étant écrit en python, nécessite pour son fonctionnement l'installation de plusieurs librairies ne faisant pas partie de la librairie standard :

- Tkinter : Utilisé pour la création de l'interface graphique

```
sudo apt-get install python3-tk
```

- CustomTkinter : Cette librairie se basant sur Tkinter permet de réaliser des interfaces plus moderne. Bien que par mes choix de "design" elle est finalement peut utilisé.

```
pip3 install customtkinter
```

- PycryptodomeX : librairie vu en cours contenant plusieurs algorithmes de chiffage

```
pip3 install pycryptdomex
```

- Pyperclip : Permet la copie dans le presse-papier

```
pip3 install pyperclip
```

Présentation du projet :

Le but de ce projet est de réaliser un gestionnaire de mot de passe permettant le stockage de nos informations de connexion de manière sécurisée, via l'utilisation d'algorithme de cryptographie et d'un mot de passe maître choisi par l'utilisateur.

Le gestionnaire doit être capable de déchiffrer les informations stocker quand le mot de passe maître lui est fourni, et permettre à l'utilisateur de les récupérer.

Un générateur de mot de passe sécurisé doit aussi être proposé.

Pourquoi le gestionnaire de mot de passe ? :

Dans mon alternance commencé la semaine suivant le cours intensif, j'ai été amené à en utiliser un pour stocker les nombreux identifiants, J'ai trouvé cela intéressant à reproduire

Pourquoi le nom K(u)eeper ? :

Ce nom est tout simplement un "jeu de mot", venant du mot anglais "Keeper"

et de la ceinture d'astéroïdes "Kuiper", les parenthèse entourant le "u" représente une chaîne d'astéroïde, je n'ai malheureusement pas les talents de graphiste nécessaire à réaliser un logo représentant le nom plus explicitement, bien que le thème de l'espace soit très présent dans le gestionnaire en lui même.

Pour exécuter le code il suffit d'utiliser la commande :

```
python3 main.py
```

Le gestionnaire est scindé en 3 fichier :

- Login.py :
- Main.py : le fichier principal gérant les interactions des deux classes précédente

Le changement s'effectue via la fonction `change_page`

Le programme se lance ainsi :

Guide d'utilisation :

Sur la page de connexion vous pouvez crée un nouveau coffre ou en importer un, dans ce cas saisissez votre mot de passe dans le champ et appuyer sur entrer ou la touche espace.

Sur la page du coffre, le bouton “Ajouter” vous permet d’ajouter un mot de passe avec coffre.

Il s'affichera sur l'emplacement à droite, une scrollbar est intégré si besoin. Pour pouvoir accéder à votre mot de passe il suffit de double-cliquer sur la cellule contenant le mot de passe souhaité, il sera copier dans le presse-papier (cela fonctionne pour tout les données stocker)

Pour supprimer un champ, il faut d'abord le sélectionner puis cliquer sur le bouton "supprimer" sinon ça ne fonctionnera pas.

Le bouton “modifier” fonctionne comme le bouton supprimer, sélectionner d’abord le champ à modifier puis cliquer sur le bouton modifier.

Le bouton "Fermer le coffre" retourne à la page de connexion.

Un coffre pré-remplie vous est fourni :

Nom : Quoi.db

Mot de passe : feur

Le projet a été testé sur 5 machines différentes(2 WSL, 1 windows, 2 linux), une seul (sur linux) a rencontré des problèmes avec la copie dans le presse-papier du mot de passe.

Dans l'éventualité où cela vous arrive aussi, voici la configuration de la machine sous linux n'ayant eu aucun soucis avec le gestionnaire:

```


→ Downloads neofetch
      ,,:,,,:,,
    ,',:cccccccccccccc:,
    ,:cccccccccccccccccccc:,
    ,:cccccccccccccccccccccc:,
    ,:cccccccccccccc:,ddd:,:cccccc:,
    ,:cccccccccccccc;OwMKOOXMMw;:cccccc:,
    ,:cccccccccccccc;KMMc;cc;xMMc:cccccc:,
    ,:cccccccccccccc;MMM.;cc;wW:;cccccccc,
    ,:cccccccccccccc;MMM.;:cccccccccccccc:,
    ,:cccccc;ox000o;MMM00ok;:cccccccccccc:,
    ,:cccccc;0MMKxdd;MMKddc;:cccccccccccc:,
    ,:cccccc;XM0';cccc;MMM.;:cccccccccccccc'
    ,:cccccc;MMo;cccc;MMW.;:cccccccccccccc;
    ,:cccccc;0Mnc.ccc.xMMd;:cccccccccccccc;
    ,:cccccc;dNMWXXXwM0;:cccccccccccccc;,
    ,:cccccccc;.:odl;.:cccccccccccccc;,,
    ,:cccccccccccccccccccccccccccccc;'.
    ,:cccccccccccccccccccccccccc;:,...
    ',:cccccccccccccc;:,,:

```

```

clem@fedora
-----
OS: Fedora Linux 38 (KDE Plasma) x86_64
Host: N350TW
Kernel: 6.5.8-200.fc38.x86_64
Uptime: 21 mins
Packages: 2409 (rpm)
Shell: zsh 5.9
Resolution: 1920x1080
DE: Plasma 5.27.8
WM: KWin
Theme: [Plasma], Breeze [GTK2/3]
Icons: [Plasma], breeze [GTK2/3]
Terminal: konsole
CPU: Intel i5-9400 (6) @ 4.100GHz
GPU: Intel CoffeeLake-S GT2 [UHD Graphics 630]
Memory: 4528MiB / 31796MiB

```



Explications du code :

Main.py :

La fonction `select_file` permet de sélectionner la base de donnée à décrypter. via la fonction `askopenfilename` de Tkinter.

La fonction `create_newDB`, permet de générer la base de données avec les champs suivants :

Table `kueeper` :

- ID : l'id du champ, il sert à nous repérer dans la BDD car les données sont cryptées. Il est utilisé dans toutes les fonctions devant interagir avec un champ spécifique de la BDD. Les 3 autres valeurs de cette table sont `site`, `utilisateur` et `mot de passe`.

Table `security` :

- Key : le hash part `sha256` du mot de passe maître servant à l'authentification.
- Primary Key : Il s'agit du mot de passe servant à crypter/décrypter les données. Il s'agit de 256 bits généré aléatoirement et crypté avec le hash part `sha3_256` du mot de passe maître.

```
def create_newDB(self):
    nom,mdp = self.get_user_input()
    print(mdp)
    conn = sqlite3.connect(nom)
    cursor = conn.cursor()
    cursor.execute("""CREATE TABLE IF NOT EXISTS kueeper (id
    INTEGER PRIMARY KEY, site TEXT NOT NULL, user TEXT NOT
    NULL, password TEXT NOT NULL)""")
    cursor.execute("""CREATE TABLE IF NOT EXISTS security (KEY
    BLOB, PRIMARY_KEY BLOB)""")
    hash_mdp = sha256(mdp.encode()).digest()
    cursor.execute("INSERT INTO security (KEY, PRIMARY_KEY)
    VALUES (?, ?)", (hash_mdp, self.gen_primary_key(mdp)))
    conn.commit()
    conn.close()
```

Pourquoi une base de données sql ? :

Certains de mes camarades ont utilisé des fichiers texte, j'ai trouvé cela contraignant et rempli de problèmes à devoir prendre en compte et corriger. SQL prend en charge la majorité de nos besoins.

La fonction open_chest :

```
def open_chest(self,entry):
    conn = sqlite3.connect(self.coffre)
    cursor = conn.cursor()
    cursor.execute("SELECT KEY FROM security")
    self.login = cursor.fetchone()
    # print(self.login)
    if sha256(entry.encode()).digest()==self.login[0]:
        conn.close()
        self.primary_key=sha3_256(entry.encode()).digest()
        self.change_page()
    else:
        self.error()
```

Fonction de connexion, elle vérifie si le hash(sha256) du mot de passe entré par l'utilisateur est égal a la valeur du champ KEY, contenant le hash(sha256) du mot de passe maître de cette base de données. Dans ce cas, un changement de page est effectué et les données afficher dans la seconde page. Sinon un popup d'erreur est affiché.

La fonction gen_primary_key :

Cette fonction génère la clef primaire, qui est utilisé pour crypter et décrypter toutes les données de compte stocker dans la base de données.

Il s'agit de 256bits généré aléatoirement avec la bibliothèque pycryptodomexe qui est ensuite crypté avec le hash par sha3_256 du mot de passe maître en utilisant l'agorithme AES.

La fonction renvoie la clé + le vecteur d'initialisation généré au début de la fonction pour ne pas avoir à le stocker dans un autre champ.

Cette methode nous permet de ne jamais avoir à sauvegarder le mot de passe entré par l'utilisateur et seulement effectuer avec, une comparaison rapide pour la connexion. Ce qui renforce la sécurité.

La fonction `gen_primary_key` :

Cette fonction décrypte la clef primaire

La fonction est construite ainsi :

- Elle commence par récupérer la clef primaire stocker dans la base de données
- Elle sépare la clé de son vecteur d'initialisation (le vecteur étant concaténé à la fin de la clef)
- Puis en valeur de retour, elle décrypte la clef avec son iv récupérer précédemment, permettant ainsi de décrypter les données de la base de données.

Cette fonction sera appelé à chaque décryptage, pour ne jamais avoir à la stocker dans une variable et seulement brièvement dans la ram, augmentant ainsi la sécurité.

```
def gen_primary_key(self,mdp):
    iv = get_random_bytes(16)
    cipher=AES.new(sha3_256(mdp.encode()).digest(),AES.MODE_OFB,iv)
    p_key = cipher.encrypt(get_random_bytes(32))
    return p_key + iv#concatenation de la clef primaire + l'iv généré aléatoirement

def decrypt_p_key(self):
    conn = sqlite3.connect(self.coffre)
    cursor = conn.cursor()
    cursor.execute("SELECT PRIMARY_KEY FROM security")
    liste = cursor.fetchone()

    p_key_cipher = liste[0]
    p_key = p_key_cipher[:-16]
    iv = p_key_cipher[-16:]

    cipher = AES.new(p_key, AES.MODE_OFB, iv)

    return cipher.decrypt(p_key)
```


La fonction encrypt :

Cette fonction permet de crypter la donnée passée en paramètre,

Elle est construite ainsi :

- Elle génère le vecteur d'initialisation, la génération aléatoire à chaque chiffrement permet la protection contre les attaques de répétitions
- Elle appelle la fonction decrypt_p_key pour obtenir la clé primaire
- Elle renvoie la version cryptée de la donnée par AES concaténée avec son iv

La fonction decrypt :

Cette fonction permet de décrypter la donnée cryptée passée en paramètre,

Elle est construite ainsi :

- Elle récupère le vecteur (les 16 derniers octets) et la donnée en les séparant
- Elle appelle la fonction decrypt_p_key pour obtenir la clé primaire
- Elle renvoie la version décryptée de la donnée chiffrée par l'algorithme AES

```
def encrypt(self, champ):
    iv = get_random_bytes(16)
    cipher = AES.new(self.app_instance.decrypt_p_key(), AES.MODE_OFB, iv)
    cipher_champ = cipher.encrypt(champ.encode())
    return cipher_champ + iv #concatenation de la clé primaire + l'iv généré aléatoirement

def decrypt(self, champ):
    iv = champ[-16:]
    cipher_champ = champ[:-16] #vu que les données déchiffrées ne seront pas toutes de la même longueur on ne peut pas écrire de manière brute comme dans les fonctions de clés primaires écrites en dessous
    decipher = AES.new(self.app_instance.decrypt_p_key(), AES.MODE_OFB, iv)
    return decipher.decrypt(cipher_champ).decode('UTF-8')
```

La fonction `add_pass` :

Cette fonction permet d'ajouter et crypter la donnée passée en paramètre, ce qui signifie que les données sont ajoutées cryptées, il n'y a donc pas besoin de boucle pour crypter les données stockées.

Elle est construite ainsi :

- Elle récupère les valeurs saisies par l'utilisateur dans un popup
- Elle insère dans la table `kueeper` les données cryptées
- Elle ajoute les valeurs dans le `treeview` pour les afficher sur la page, cependant pour garantir l'intégrité des mots de passe, ils ne sont jamais affichés ni stockés dans le `treeview`, à la place un nombre de "*" équivalent à la longueur du mot de passe est affichés et stockés.

Si une erreur se produit pendant l'ajout du mot de passe (valeurs vides, etc...) un popup d'erreur sera affiché (défini dans la fonction de popup en plus d'une exception)

```
def add_pass(self,conn):
try:
site,user,password=self.app_instance.get_user_input_add_pass()
cursor = conn.cursor()
cursor.execute("INSERT INTO kueeper (site, user, password) VALUES (?, ?, ?)",
(self.encrypt(site), self.encrypt(user), self.encrypt(password)))
conn.commit()
cache= '*' * len(password)
self.tree.insert("", 'end', values=(site, user, cache,cursor.lastrowid))
except Exception as error:
print(f"Erreur lors de l'ajout du mot de passe : {error}")
```

Pourquoi l'algorithme AES ? :

Pour ce gestionnaire, de par sa nature, je devais choisir quel algorithme de chiffrement symétrique utilisé, j'ai donc opté pour l'algorithme AES, cette algorithme largement reconnu pour sa sécurité et son efficacité, il s'agit donc d'un bon choix pour une application devant chiffrer et déchiffrer régulièrement comme un gestionnaire de mots de passe. De plus j'ai choisi de l'utiliser avec le mode OFB, ce mode est efficace dans sa sécurité mais possède tout de même quelques failles, notamment l'absence d'authentification que possède le mode GCM ou CCM.