Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/mp

Méthodologie de la programmation

TP 3 : Messages secrets

Dans ce TP :
- Apprendre à utiliser les environnements virtuels de Python.
- Apprendre à utiliser le gestionnaire de paquet de Python.
- Jouer avec la bibliothèque cryptographique NaCl.

Exercice 0.

Utilisation de `venv` et `pip`.
`https://docs.python.org/fr/3/library/venv.html`
`https://docs.python.org/fr/3/installing/`

1. L'objectif de ce TP est d'écrire un outils simple de chiffrement et déchiffrement de fichiers.

   On va pour cela avoir besoin de la bibliothèques Python NaCl.

   Dans le cadre d'un vrai projet, on pourrait avoir besoin de beaucoup plus de bibliothèques, et parfois de versions spécifiques de certaines d'entre elles, en cas d'incompatibilité suite à des mises à jour.

   Bien sûr, d'une part on travaille rarement sur un seul projet à la fois, et d'autre part on souhaite pouvoir redéployer rapidement un environnement similaire (avec les mêmes bibliothèques dans les mêmes versions) ailleurs que sur sa machine de développement.

   Pour cela on utiliser un environnement virtuel.

   Python est livré avec un module `venv` qui sert justement à ça. Parfois on peut l'utiliser directement avec la commande `venv`, parfois il faut appeler la commande `python3` et lui indiquer d'utiliser le module `venv` avec l'option `-m` sur la ligne de commande. Cela revient au même.

   Dans tous les cas, on ajoute en argument un dossier dans lequel sera créé l'environnement virtuel.

   → Créez un environnement virtuel `tp3`.
2. → Rendez-vous maintenant dans le dossier nouvellement créé `tp3`.
3. Pour activer l'environnement virtuel il faut "sourcer" le script `bin/activate` en le passant en argument à la commande `source` du shell (qui va simplement interpréter les commandes shell contenue dans le script comme si vous les aviez tapé sur votre ligne de commande).

   → Activez votre environnement.
4. → Que remarquez-vous ?
5. On va maintenant installer le paquet PyNaCl (`https://pypi.org/project/PyNaCl/`) dans notre environnement virtuel.

   → Lancez la commande `pip install pynacl`, et essayez de comprendre ce qu'elle vous affiche. Quels paquets sont installés ?
6. Pour vérifier que l'installation s'est bien passée, vous pouvez maintenant ouvrir un interpréteur Python (toujours dans votre environnement virtuel !) et taper `import nacl`. Si tout s'est bien passé, il n'y aura pas d'erreur.
7. Pour sortir de votre environnement, vous pouvez utiliser la commande `deactivate`.

   → Pour l'instant, explorez un peu les fichiers et dossiers créés pour comprendre ce que contient l'environnement virtuel.

Exercice 1.

Écriture de notre script de chiffrement de fichiers.

1. → Dans votre environnement, créez un dossier `msgscrt`.
2. Dans ce dossier on va créer un script `chiffre.py`, qui permettra de chiffrer un document avec une phrase de passe.

   Nous allons utiliser pour cela une `SecretBox` du module `nacl.secret`.

   L'algorithme de chiffrement utilisé par `SecretBox` exige une clef de chiffrement d'une taille fixe de 32 octets. Il va donc nous falloir une moyen de transformer la phrase de passe donnée par l'utilisateurice en une suite de 32 octets.

   Pour cela, on va utiliser une *fonction de hachage*, c'est à dire une fonction qui prend une chaîne de caractères de longueur quelconque en entrée et qui va générer de manière déterministe une chaîne de longueur fixe en fonction de l'entrée.

La fonction de hachage que l'on va utiliser est la fonction `sha256` du module `nacl.hash`, qui renvoie une valeur sur 256 bits, c'est à dire 32 octets.

→ Créez le fichier `chiffre.py` et importez dedans les deux modules du paquet `nacl` que l'on va utiliser.

3. On va commencer par écrire une fonction `secret_phrase` qui demande une phrase de passe à l'utilisateurice et qui la transforme en tableau de 32 octets.

   (a) → Utilisez la fonction `input` pour demander une phrase de passe à l'utilisateurice et stockez la dans une variable `passphrase`.

   (b) La fonction `nacl.hash.sha256` prend en argument un tableau d'octets, il nous faut donc convertir la phrase de passe donnée par l'utilisateurice, on peut le faire avec la méthode `encode` de la classe `str`. Cette méthode prend en paramètre l'*encodage* de la chaîne de caractères, qui pour nous est `utf-8`.

   → Faites renvoyer à la fonction le résultat de l'appel à la fonction `nacl.hash.sha256` auquel vous passez en argument `passphrase.encode('utf-8')`, ainsi que `encoder=nacl.encoding.RawEncoder` pour qu'elle nous renvoie directement le tableau d'octets.

   (c) → Pensez à tester votre code !

4. Nous allons maintenant écrire la fonction `encrypt_file` qui chiffre un fichier dont le chemin est donné en argument (`path`), en utilisant une clef qui lui est également donnée en argument (`key`). Commençons par la lecture du contenu du fichier.

   (a) Pour lire le contenu d'un fichier, il faut d'abord l'ouvrir en mode lecture avec la fonction `open`. Cette fonction prend en premier argument le chemin vers le fichier et en second argument le *mode* d'ouverture du fichier. Pour nous le mode sera `"rb"` : "r" pour "read" et "b" pour "binary" car on lit le fichier en mode binaire pour récupérer directement ses données brutes sous forme d'un tableau d'octets (car le fichier ne contient pas nécessairement du texte).

   → Ouvrez le fichier à chiffrer en stockez le résultat de l'ouverture dans une variable `f`.

   (b) Votre variable `f` contient maintenant un objet "fichier" de Python. Vous pouvez utiliser dessus la méthode `read` qui vous renverra le contenu du fichier (elle ne prend aucun argument).

   → Stockez le contenu du fichier dans une variable `content` : `content = f.read()`.

   (c) → Vous pouvez ensuite fermer le fichier avec la méthode `close` (qui ne prend aucun argument non plus).

5. On va maintenant effectuer le chiffrement du fichier grâce à la `SecretBox` de NaCl.

   (a) → Affectez à la variable `box` une instance de `SecretBox` qui utilise la clef qu'on a reçu en argument dans la fonction `encrypt_file` avec le code suivant : `box = nacl.secret.SecretBox(key)`.

   (b) Pour chiffrer le contenu du fichier, on le passe en argument à la méthode `encrypt` de notre instance de `SecretBox`.

   → Affectez à la variable `encrypted` la version chiffrée du contenu de notre fichier.

6. Nous allons maintenant écrire la version chiffrée du fichier dans un nouveau fichier, dont le nom sera celui du fichier original, auquel vous aurez ajoutez ".encrypted" (pour éviter d'écraser votre fichier existant).

   Pour ajouter un suffix à une chaîne de caractère vous pouvez utiliser l'opérateur `+`.

   Le mode d'ouverture d'un fichier en mode binaire pour l'écriture est `"wb"` (avec un "w" pour "write" cette fois-ci). Attention cela va écraser le contenu du fichier s'il existe !

   La méthode pour écrire un contenu dans un fichier ouvert en mode écriture est `write`, qui prend ledit contenu en argument.

   → Ouvrez, écrivez, puis fermez la version chiffrée du fichier.

7. On veut maintenant que notre script soit utilisable depuis la ligne de commande.

   → Comment vérifie-t-on que notre script est utilisé comme programme principal et pas importé comme module ?

8. Dans le cas où notre script est utilisé comme programme principal, nous avons besoin que l'utilisateurice ait fourni le nom du fichier à chiffrer en argument sur la ligne de commande, par exemple : `python3 chiffre.py message_secret.txt`.

   Pour cela nous allons importer le module `sys` de la bibliothèque standard de Python, qui nous permet d'accéder au tableau `sys.argv` qui contient les arguments de la ligne de commande.

   La première case du tableau, `sys.argv[0]` contient le nom du programme, et les cases suivantes les arguments. Dans le cas de l'exemple ci-dessus, la valeur de `sys.argv[1]` serait `"message_secret.txt"`.

   La fonction `len` renvoie la longueur du tableau que vous lui passez en argument.

   → Vérifiez que l'utilisateurice a bien donné un nom de fichier en argument, et si non, lui signaler une erreur et quitter le programme avec `sys.exit(1)`.

9. → Si un nom de fichier a bien été donné, demandez à l'utilisateurice d'entrer une phrase de passe à l'aide de notre fonction `secret_phrase` et appelez ensuite la fonction `encrypt_file` pour générer la version chiffrée du fichier, et annoncez que c'est le cas à l'utilisateurice.
10. → Testez votre code !

Exercice 2.

Écriture de notre script de déchiffrement de fichiers.

1. Pour le script de déchiffrement, on va réutiliser la fonction `secret_phrase` du module `chiffre.py`.
   On aura en revanche besoin d'écrire une fonction `decrypt_file` qui utilisera aussi une `SecretBox` du module `nacl.secret`
   → Créez un fichier `dechiffre.py` dans votre dossier `msgscrt` et importez y le module `nacl.secret` et la fonction `secret_phrase` du module `chiffre`.
2. → Écrivez la fonction `decrypt_file(path, key)` qui fait la même chose que son homologue de chiffrement mais appelle la méthode `decrypt` de la `SecretBox` et écrit le résultat dans un fichier avec le suffixe `.decrypted`.
3. On veut permettre à l'utilisateurice de déchiffrer des fichiers depuis la ligne de commande avec la même interface d'utilisation que pour le script `chiffre.py`.
   → Écrivez ce que le script doit faire si il est appelé comme programme principal.

Exercice 3.

Transformation de nos scripts en paquet Python.

1. Notre objectif est maintenant de transformer notre dossier `msgscrt` en paquet Python, de sorte à ce que nos script puissent être appelés avec la commande `python3 -m msgscrt.chiffre`, par exemple.
   → Pour cela, créez un fichier `__init__.py` dans le dossier.
2. Testez vos deux scripts en remontant dans le répertoire racine de votre environnement virtuel (celui où vous avez créé le dossier `msgscrt`).
   → Quel soucis constatez-vous ?
3. Il faut en effet utiliser un *chemin d'import relatif* pour le module `chiffre` dans le script `dechiffre.py`.
   → Corrigez l'erreur et vérifiez que tout fonctionne dorénavant correctement.
4. → Pour rendre votre paquet installable localement créez un fichier `setup.py` à côté du dossier `msgscrt` avec le contenu suivant :

```python
from distutils.core import setup

setup(name='msgscrt', version='0.0', packages=['msgscrt'])
```

5. → Pour installer votre paquet msgscrt dans votre environnement virtuel, utilisez la commande `pip install .` (on rappelle que le `.` représente le dossier courant, indiquant à `pip` où trouver le fichier `setup.py`).
6. Vous pouvez maintenant utiliser votre paquet Python depuis n'importe où du moment que l'environnement virtuel dans lequel il est installé est activé !
   → Testez d'utiliser `python3 -m msgscrt.chiffre` et `python3 -m msgscrt.dechiffre` (et que les opérations de chiffrement et déchiffrement fonctionnent effectivement) par exemple depuis votre `$HOME`.
7. Pour désinstaller et réinstaller votre paquet (par exemple pour prendre en compte des mises à jour de votre code) vous pouvez faire `pip uninstall msgscrt` puis de nouveau la commande pour le réinstaller.
   Vous pouvez aussi l'installer en mode développement pour que les changements du code du paquets soient immédiatement pris en compte en passant l'option `-e` (pour "editable") à la commande `install` de `pip`.