Série 2

Maintenant que vous connaisez un peu le Jupyter, ça devrait être plus simple. Dans cette série, nous allons découvrir les functions de hachage. On va d'abord faire quelques essais avec des functions de hacahge peu performant, pour voire comment on peut les attaquer. Ensuite on va passer aux *vraies* functions de hachage, et voir que les attaques sont beaucoup plus difficiles que ça.

Exercice 2

On va maintenant s'intéresser à une vraie fonction de hachage, le `sha-256`. Cette fonction a été testé, a été attaqué en vein, et est utilisé partout dans le monde. On va donc considérer que c'est une fonction de hachage cryptographique.

Néanmoins, même si on prend une chose quasi-parfaite, en l'utilisant d'une manière inadéquate, on peut quand même se tromper. Ici on verra quelques exemples.

Ici on suppose qu'un serveur a stocké les hachage des mots de passe de ses utilisateurs. On va voir comment un hacker peut récupérer les mots de passe, même si théoriquement c'est impossible.

1. Connaissance

Si vous faites tourner le block pour la partie 1, vous verrez que le hachage `sha256` retourne une valeur beaucoup plus longue que notre fonction simple. En plus, la valeur retournée par le `sha256` a une longueure constante. Mais tout ceci ne nous aide pas si on ne fait pas attention.

Une autre chose intéressante est l'utilisation de la fonction `sha256`. Ligne par ligne, la chose suivante se passe:

- `sha = sha256()` crée un objet qui permet d'utiliser le hachage
- `sha.update(phrase.encode())` fournit la phrase à notre nouvel objet. Une phrase en informatique peut être écrite de différente manière, alors ici on doit spécifier qu'on suppose la plus simple avec `encode()`. Un autre point intéressant est le fait que le `update` peut être appelé plusieurs fois. Mais on ne va pas le faire ici.
- `sha.hexdigest()` finalise le hachage et fournit le résultat final dans un joli texte hexadécimal

Changez les phrases et prenez soin de bien vérifier qu'un tout petit changement de la phrase change complètement le résultat du hachage.

2. Compréhension

Supposons que le serveur ait stocké le hachage suivant pour un mot de passe:

```
88ec4a1bbddb2ac8442d6c5b443c4d5978b544362ead2590d19b8b1e7e27fea6
```

à priori il est quasiment impossible de trouver quel est le mot de passe qui a généré ce hachage. Par contre si vous savez que c'est un mot de passe qui a été créé par une combinaison des mots suivants, c'est possible:

```
secret password nobody knows
```

Essayez de "cracker" le mot de passe en testant la combinaison de ces quatre mots, jusqu'à trouver le mot de passe correcte.

3. Application

On va essayer d'écrire un petit cracker de mot de passe. Vu que c'est très lent, on va se limiter aux mots de passes suivants:

- lettres minuscules a-z
- longueur: 2

complétez la fonction "hack_pass" pour trouver les mots de passe correspondant aux hachages suivants:

```
961b6dd3ede3cb8ecbaacbd68de040cd78eb2ed5889130cceb4c49268ea4d506
970f519c2cadbcefb1e81694f904bc6229dd2a8300e98c6d0d4fc4bfca584140
4a60bf7d4bc1e485744cf7e8d0860524752fca1ce42331be7c439fd23043f151
8fa1dddd53606ceb933c5c6a12e714ed41e11d37a2b7bc48e91d15b54171d033
```

In [2]:

```
# Exercice 2 - Partie 1

from hashlib import sha256

def sha256_str(phrase: str) -> str:
  sha = sha256()
  sha.update(phrase.encode())
  return sha.hexdigest()

def print_sha256(phrase: str) -> str:
  print('sha256("{}") is: {}'.format(phrase, sha256_str(phrase)))

print_sha256("secret password")
print_sha256("nobody knows")
```

```
sha256("secret password") is: 1ba133eccdfc4e5ca3405dfd70c11360af038106c9eebdde504a4b14c94b8557
sha256("nobody knows") is: 03982e2adb2925b16d9a1a591a0665a8db33153c21d9fc1895a016da460d35cd
```

In [8]:

```
# Exercice 2 - Partie 2

def check_password(hash: str, password: str):
  if sha256_str(password) == hash:
    print("Please enter into our system")
  else:
    print("Disappear evil hacker and never come back again")

check_password("88ec4a1bbddb2ac8442d6c5b443c4d5978b544362ead2590d19b8b1e7e27fea6",
               "nobody password")
```

```
Please enter into our system
```

In [11]:

```
# Exercice 2 - Partie 3
import string

def verify_hack(hash: str):
    check_password(hash, hack_pass(hash))

def hack_pass(hash: str) -> str:
    for a in string.ascii_lowercase:
        for b in string.ascii_lowercase:
            if sha256_str(a + b) == hash:
                print("Found password:", a + b)
                return a + b

    return "unknown password"

verify_hack("961b6dd3ede3cb8ecbaacbd68de040cd78eb2ed5889130cceb4c49268ea4d506")
verify_hack("970f519c2cadbcefb1e81694f904bc6229dd2a8300e98c6d0d4fc4bfca584140")
verify_hack("4a60bf7d4bc1e485744cf7e8d0860524752fca1ce42331be7c439fd23043f151")
verify_hack("8fa1dddd53606ceb933c5c6a12e714ed41e11d37a2b7bc48e91d15b54171d033")
```

```
Found password: aa
Please enter into our system
Found password: ba
Please enter into our system
Found password: zz
Please enter into our system
Found password: mm
Please enter into our system
```

In [ ]: