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 3

Dans ce troisième exercice on va voir comment un tableau peut faire une attaque beaucoup plus rapide, mais comment du sel peut faire que le tableau perd tous ses effets.

1. Connaissance

Dans la partie application de l'exercice 2 on a demandé de faire une fonction qui fait du "brute force" pour trouver un mot de passe. Ceci est en fait assez lent, vu qu'on doit faire tout le travail pour chaque nouveau mot de passe.

Une version beaucoup plus rapide est le tableau précalculé. Il y a différentes version, les plus élaborées peuvent prendre un bon semestre pour bien les expliquer. Ici on va se contenter de calculer le tableau une fois, et puis de l'appliquer aux hachages qui vont suivre.

Comme dans la partie 3 de l'exercice 2, on va se contenter de limiter les mots de passe, mais ici on les fait un peu plus long: cinq lettres minuscules de a-z. Pour des raisons de limitations de mybinder.org, la dernière lettre ne peut comporter que de a-j.

La partie ingénieuse vient du fait que l'initialisation doit être calculé seulement une fois. Ici on n'a que 10 secondes pour la création, mais avec plus de lettres, ça peut prendre longtemps.

Après on peut faire tourner la partie 1b pour cracker quelques mots de passe.

2. Compréhension

Pour éviter ce problème, pour chaque mot de passe on crée un `sel` qui n'est rien d'autre qu'une chaîne de lettres aléatoires. Ce sel permet de faire le travail du cracker beaucoup plus difficile.

Lancez la partie 2 plusieurs fois, et vous verrez les choses suivantes:

- le sel change à chaque fois
- le hachage aussi change
- la vérification du mot de passe réussit quand même

Pour cet exercice, ajoutez une deuxième création de mot de passe avec sel, puis vérifiez que les deux hachages salés sont reconnus comme représentant le mot de passe.

3. Application

Maintenant vous pouvez essayer de craquer le mot de passe salé avec notre fonction de la partie 1:

- créer une entrée de hachage avec sel pour un mot de passe avec 5 lettres
- séparez le hachage d'avec le sel
- essayez de récupérer le mot de passe en passant le hachage dans la fonction `crack_password`

In [1]:

```
# Exercice 2 - Partie 1a - Initialization - Run this only once!
import string, time
from hashlib import sha256

# Returns the hexadecimal string of the sha256
def sha256_str(phrase: str) -> str:
  sha = sha256()
  sha.update(phrase.encode())
  return sha.hexdigest()

# Pretty-prints a phrase and its sha256 representation
def print_sha256(phrase: str) -> str:
  print('sha256("{}") is: {}'.format(phrase, sha256_str(phrase)))

# Will contain all passwords and their hash
password_table = {}

def create_table():
  for a in string.ascii_lowercase:
    for b in string.ascii_lowercase:
      for c in string.ascii_lowercase:
        for d in string.ascii_lowercase:
          for e in string.ascii_lowercase[0:10]:
            pwd = a + b + c + d + e
            password_table[sha256_str(pwd)] = pwd

# Tries to look up the password in the table
def crack_password(hash: str) -> str:
  pwd = password_table.get(hash)
  if pwd != None:
    print('Found password "{}" corresponding to hash "{}"'.format(pwd, hash))
  else:
    print('Sorry, didn\'t find password corresponding to hash "{}"'.format(hash))

print("Creating table - please wait")
start = time.time()
create_table()
print("Creation of table finished. Time elapsed: {}".format(time.time() - start))
```

```
Creating table - please wait
Creation of table finished. Time elapsed: 10.179802894592285
```

In [2]:

```
# Exercice 2 - Partie 1b - Usage

crack_password("ed968e840d10d2d313a870bc131a4e2c311d7ad09bdf32b3418147221f51a6e2")
crack_password("36bbe50ed96841d10443bcb670d6554f0a34b761be67ec9c4a8ad2c0c44ca42c")
# The following is zzzzz, which is not in the table!
crack_password("68a55e5b1e43c67f4ef34065a86c4c583f532ae8e3cda7e36cc79b611802ac07")
crack_password("3bd8a5b54baf23f33a1ebe80a5b7339da4b4d0c2c4a6918d80f3b277ae5bdcfa")
```

```
Found password "aaaaa" corresponding to hash "ed968e840d10d2d313a870bc131a4e2c311d7ad0
9bdf32b3418147221f51a6e2"
Found password "abcde" corresponding to hash "36bbe50ed96841d10443bcb670d6554f0a34b761
be67ec9c4a8ad2c0c44ca42c"
Sorry, didn't find password corresponding to hash "68a55e5b1e43c67f4ef34065a86c4c583f5
32ae8e3cda7e36cc79b611802ac07"
Found password "zzzza" corresponding to hash "3bd8a5b54baf23f33a1ebe80a5b7339da4b4d0c2
c4a6918d80f3b277ae5bdcfa"
```

In [17]:

```
# Exercice 2 - Partie 2
import random

def salt(length: int) -> str:
    return ''.join(random.choice(string.ascii_lowercase) for i in range(length))

def create_salted_password(password: str) -> str:
    pwd_salt = salt(8)
    hash = sha256_str(pwd_salt + password)
    return "{}::{}".format(pwd_salt, hash)

def check_salted_password(salted_entry: str, password: str):
    pwd_salt, hash = salted_entry.split("::")
    if hash == sha256_str(pwd_salt + password):
        print("Success! Password matches")
    else:
        print("Failed - bad hacker")

password_entry = create_salted_password("abcde")
print('Salted entry for password "abcde" is: {}'.format(password_entry))
check_salted_password(password_entry, "abcde")
```

```
Salted entry for password "abcde" is: ryntjlhz::b23f5b5cc1a293f809d3a2dd441a6fe072630d
87f95e42cb375a3ff45784a710
Success! Password matches
```

In [23]:

```
# Exercice 2 - Partie 3

[pwd_salt, pwd_hash] = create_salted_password("abcde").split("::")
print('Password hash is: {}, salt is: {}'.format(pwd_hash, salt))
cracked = crack_password(pwd_hash)
```

```
Password hash is: b5e21d90b91ae31bdb5a78e278209d5da2f2dd7cfb20cc3a184dd4bcc232baec, sa
lt is: <function salt at 0x13f535b80>
Sorry, didn't find password corresponding to hash "b5e21d90b91ae31bdb5a78e278209d5da2f
2dd7cfb20cc3a184dd4bcc232baec"
```