# Hands-on Brute Force Attack

# Python and Cryptography package

- I assume each of you have a python distribution installed
- In Windows I have used the package Cryptography (and installed with `pip install cryptography`)
- For instructions on the packages you may look at https://pypi.org/
- Install the package with **`python pip install name_package`**:
  - **`pip install cryptography`**
- Note that the package has many functionalities, we will see only very few of them

# Using the Cryptography package

```
import base64
import os
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

passwd = b"XXXXXXX"
cleartext = b"this is an example of plaintext to encrypt and decrypt"
salt = os.urandom(16)
print('Salt = ',salt)
kdf = PBKDF2HMAC(
     algorithm=hashes.SHA256(),
     length=32,
     salt=salt,
     iterations=100000,
)
key = base64.urlsafe_b64encode(kdf.derive(passwd))
f = Fernet(key)
cyphertext = f.encrypt(cleartext)
print('Cyphertext = ',cyphertext)
print(f.decrypt(cyphertext))
```

The Base64 encoding is used to convert bytes that have binary or text data into ASCII characters. Encoding prevents the data from getting corrupted when it is transferred or processed through a text-only system.

Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

PBKDF2 (Password Based Key Derivation Function 2) is typically used for deriving a cryptographic key from a password. Here it sets its parameters.

Derives the key from the password and sets the key in the library

Encrypts and decrypts

## Notes

- The code uses a human-readable password to generate a key
  - Typically by hashing the password a number of times
- However that’s not always the case…
  - If you have already the key you just encrypt

## Decrypting…

```
import base64
import os
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

def decrypt(passwd):
   salt = b"\x............" # you should write here the salt obtained when
encrypting
   cyphertext =  b'............' # you should write here the cyphertext obtained
when encrypting
   kdf = PBKDF2HMAC(
      algorithm=hashes.SHA256(),
      length=32,
      salt=salt,
      iterations=100000,
   )
   key = base64.urlsafe_b64encode(kdf.derive(passwd))
   f = Fernet(key)
   try:
      print(f.decrypt(cyphertext))
      print('right password: '+str(passwd)+'\n')
   except:
      print('wrong password: '+str(passwd))
      pass

decrypt(b"TestPass")
```

## Notes

The code is a fragment of what an attacker should do:

- He has a salt and a cyphertext
- He generate one password at a time…
- … and he tryes to decrypt

Why does the try-except  discovers an attempt to decrypt failed?

# Exercise

You have intercepted a cyphertext (and the corresponding salt) that you wish to decrypt:

You suspect the sender has used a password in a dictionary (for simplicity the one on the left-hand side…), possibly with one character in capital letters and combining the password with a number and a special symbol to lightly satisfy the requirement of a complex password policy.

For example, the string "qwerty" may be combined with a number and a special character to obtain passwords such as "qw3ert#Y" or "@Qwerty4".

Hence you decide to attempt with a combination of dictionary and brute force attack to test all the possible passwords. Once decrypted the cyphertext you will have also discovered the password.

Assume there are 10 special characters available:

! $ % & ? ^ * + @ #

| Dictionary |
|---|
| gatto |
| giulia |
| martina |
| password |
| pisa |
| poesia |
| qwerty |
| sicurezza |
| storia |
| tavolo |

# Exercise

Hints:

The combinations are many. Before attempting the brute force attack I suggest you to test your code on a small scale, for example creating a simple password and checking on that your code is able to find the combination.

To attack the full problem instead:

- determine first the number of different password you need to generate to implement your attack against the cyphertext
- Assess the time that may take to check all the passwords
  - Hint: measure how long it takes to check (let’s say) 1000 passwords on your PC …
  - Hint: generation of passwords can be very slow, avoid the use of too many prints in the code and see what happens…
- Estimate the worst case and average-case time your password cracker will take

The cyphertext and the salt are:

```
salt =  b"\xd2\xffs~\xb4\xf2\xd3\xda\xe3\x16('\xe6\xad\xef\xaf"

cyphertext =  b'gAAAAABnE2P-qqJT-HudMbLcykzIx83XqZNEt6UqfyBBzhYKvlF9WSx8FJUvUmatzuY1-
io9RHWaj7RVBuAKTWRAVT9GpGC--
TZUXk387qeTC2jIJOfUrwSX3eGEb1EVFZBOqALd8EKS1CFWUoF4NpzKsc3eLeCnXihb-w6Boqi835uNzN6mZz4iP-
6sSkhNxHP-TbrG-BNgjMIyeRDjSLAZhEAJoUGlz_QuOyyOYHMab9LUrXkHibU='
```

(courtesy of your colleague G. Lombardi)

## Solution

[redacted]

## Solution

## Solution