CYBER CHALLENGE.IT

Cryptography 1

Carmen Casulli & Fabio Giovanazzi

21/03/2024

UNIVERSITÀ DI TRENTO

© Carmen Casulli, Fabio Giovanazzi

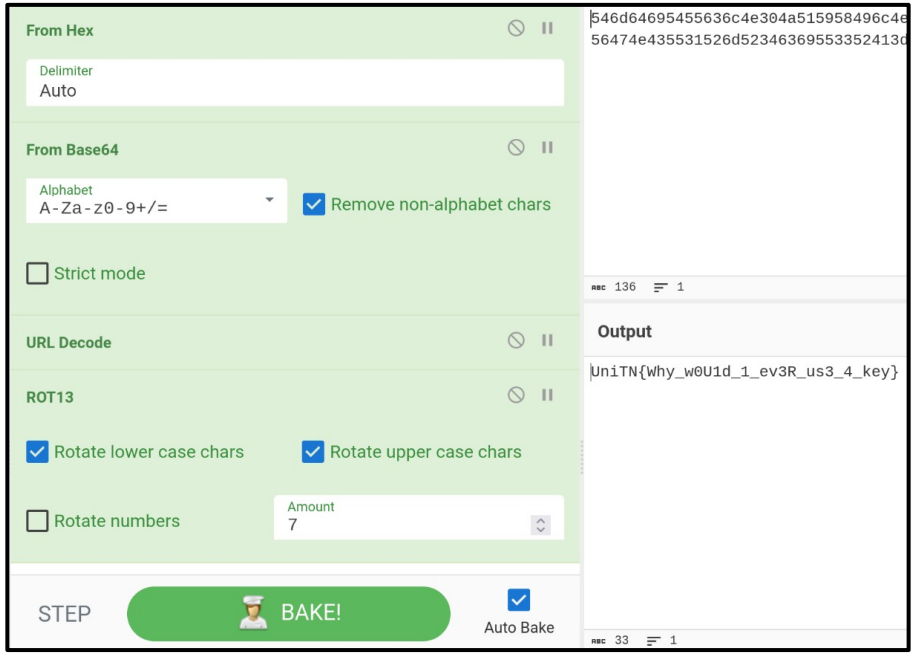
Challenge 1: Secret encoding

Please deliver this message to Alice as soon as possible. Don't bother trying to read it, I protected it with advanced enco..encryption!

```
546d64695455636c4e304a515958496c4e
555a774d4534786479553152 6a456c4e55
5a34627a4e4c4a545647626d777a4a5456
474e435531526d52346369553352413d3d
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 1 solution



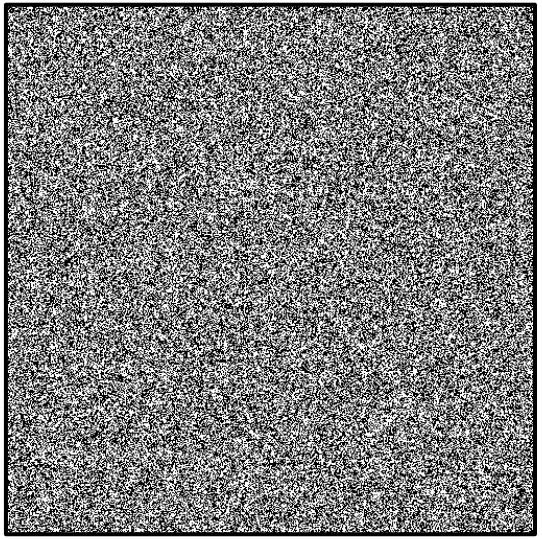
Use CyberChef!

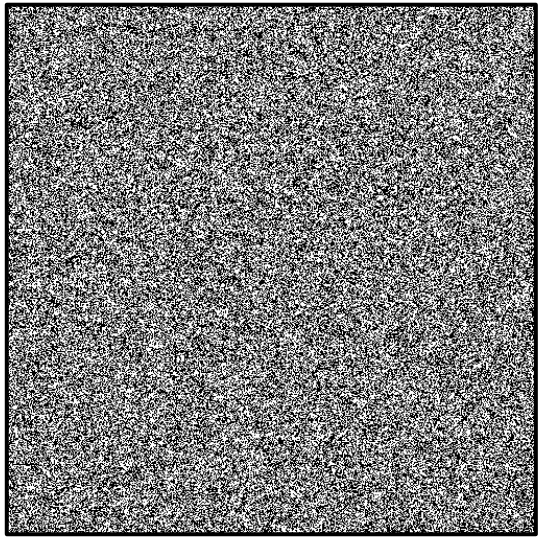
1. base16/HEX
2. base64
3. URL/percent encoding
4. ROT 7

© Carmen Casulli, Fabio Giovanazzi

Challenge 2: Random pictures

Encoding wasn't secure enough for the secret conversations between me and Alice… Now we agreed on a single key, so we can perform real encryption with XOR!





Challenge 2: Random pictures

Encoding wasn't secure enough for the secret conversations between me and Alice… Now we agreed on a single key, so we can perform real encryption with XOR!

Hints:

- The images have been XORred with the same key

© Carmen Casulli, Fabio Giovanazzi

Challenge 2: Random pictures

Encoding wasn't secure enough for the secret conversations between me and Alice… Now we agreed on a single key, so we can perform real encryption with XOR!

Hints:

- The images have been XORred with the same key
- The challenge can be solved with a single line of code

© Carmen Casulli, Fabio Giovanazzi

Challenge 2 solution

- Both images have been XORred with k:

```
flag = orig_flag ^ k
peppers = orig_peppers ^ k
⇒ flag^peppers = orig_flag ^ orig_peppers (^ k ^ k = ^ 0)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 2 solution

- Both images have been XORred with k:

```
flag = orig_flag ^ k
peppers = orig_peppers ^ k
⇒ flag^peppers = orig_flag ^ orig_peppers (^ k ^ k = ^ 0)
```

- We can XOR the images with **imagemagick** from command line

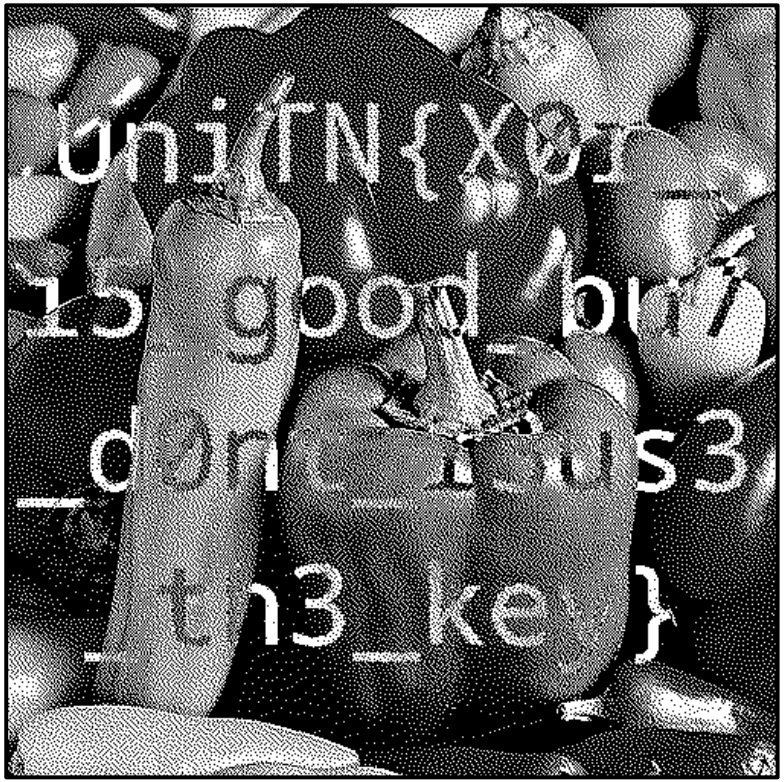
```
magick flag.png peppers.png -evaluate-sequence xor result.png
```

- Or in Python we can use `PIL.ImageChops.logical_xor()`

- Or many other methods…

© Carmen Casulli, Fabio Giovanazzi

Challenge 2 solution



© Carmen Casulli, Fabio Giovanazzi

pwntools

- Install the **pwntools** Python package and import it

```
import pwn
```


© Carmen Casulli, Fabio Giovanazzi

pwntools

- Install the **pwntools** Python package and import it

```
import pwn
```

- Connect to the challenge (to a remote IP, or use a local command to test the challenge locally):

```
conn = pwn.remote('ip_address', port)
conn = pwn.process(['python3', 'challenge.py'])
```


© Carmen Casulli, Fabio Giovanazzi

pwntools

- Install the **pwntools** Python package and import it

```
import pwn
```

- Connect to the challenge (to a remote IP, or use a local command to test the challenge locally):

```
conn = pwn.remote('ip_address', port)
conn = pwn.process(['python3', 'challenge.py'])
```

- Receive output:

```
conn.recvall(timeout=1)
conn.recvuntil(b'something')
conn.recvline()
```


© Carmen Casulli, Fabio Giovanazzi

pwntools

- Install the **pwntools** Python package and import it

```
import pwn
```

- Connect to the challenge (to a remote IP, or use a local command to test the challenge locally):

```
conn = pwn.remote('ip_address', port)
conn = pwn.process(['python3', 'challenge.py'])
```

- Receive output:

```
conn.recvall(timeout=1)
conn.recvuntil(b'something')
conn.recvline()
```

- Send bytes:

```
conn.sendlineafter(b'something', b'your_bytes')
```

PyCryptodome

- Install the **pycryptodome** Python package (*not* pycryptodomex, otherwise you will import from Cryptodome instead of Crypto)

© Carmen Casulli, Fabio Giovanazzi

PyCryptodome

- Install the **pycryptodome** Python package (*not* pycryptodomex, otherwise you will import from Cryptodome instead of Crypto)

- Import a cipher (e.g. AES)

```
from Crypto.Cipher import AES
```


© Carmen Casulli, Fabio Giovanazzi

PyCryptodome

- Install the **pycryptodome** Python package (*not* pycryptodomex, otherwise you will import from Cryptodome instead of Crypto)

- Import a cipher (e.g. AES)

```
from Crypto.Cipher import AES
```

- Create a new cipher

```
aes = AES.new(key, AES.MODE_ECB)
```


© Carmen Casulli, Fabio Giovanazzi

PyCryptodome

- Install the **pycryptodome** Python package (*not* pycryptodomex, otherwise you will import from Cryptodome instead of Crypto)

- Import a cipher (e.g. AES)

```
from Crypto.Cipher import AES
```

- Create a new cipher

```
aes = AES.new(key, AES.MODE_ECB)
```

- Encrypt and decrypt:

```
ciphertext = aes.encrypt(plaintext)
plaintext = aes.decrypt(ciphertext)
```

Challenge 3: pwntools and PyCryptodome

Now it's your turn to use pwntools and PyCryptodome!

```
print("Here is a key (hex):", key.hex())
print("Here is a message (hex):", message.hex())

input("What is the AES encrypted message using the given key (hex)? ")
input("What is the the MD5 hash digest of the message (hex)? ")

print(FLAG)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 3 solution

```python
import pwn
from Crypto.Cipher import AES
from Crypto.Hash import MD5

# connect to the remote shell, this is the equivalent of:
#   nc cyberchallenge.disi.unitn.it 10003
r = pwn.remote("cyberchallenge.disi.unitn.it", 10003)

# use this instead, if you want to run the challenge locally and test there
r = pwn.process(["python3", "challenge.py"])
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 3 solution

```
# receive and discard everything until ": "
r.recvuntil(b": ")
# read the line and drop the \n, decode the received bytes to string,
# then obtain bytes from the hex-encoded string
key = bytes.fromhex(r.recvline(keepends=False).decode())
r.recvuntil(b": ")
message = bytes.fromhex(r.recvline(keepends=False).decode())

# construct an AES cipher based on the given key
aes = AES.new(key, AES.MODE_ECB)
# use the cipher to encrypt the message, then obtain the corresponding
# hex-encoded string
encrypted = aes.encrypt(message).hex()
# encode the encrypted string to bytes and send it after "? " is received
r.sendlineafter(b"? ", encrypted.encode())
```

Challenge 3 solution

```
# construct an MD5 hash, and hash the message
md5 = MD5.new(message)
# obtain the hex digest from the hash object
digest = md5.hexdigest()
# encode the digest to bytes and send it after "? " is received
r.sendlineafter(b"? ", digest.encode())

# print the flag!
print(r.recvline())
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 4: EncMachinery™

Our EncMachinery™ algorithm is so secure that we even let you look at encrypted company secrets...

```
cipher = AES.new(key, AES.MODE_ECB)

def encrypt(m: bytes) -> bytes:
    # a bit of encoding to make hackerz more confused
    for _ in range(3):
        m = m.hex().encode("utf-8")
    m = pad(m, BLOCK_SIZE)
    return cipher.encrypt(m)

print("Our encrypted company secrets: " + encrypt(FLAG).hex())
```

Challenge 4: EncMachinery™

Our EncMachinery™ algorithm is so secure that we even let you look at encrypted company secrets…

Hints:

- Take a look at the **ECB** block cipher mode

© Carmen Casulli, Fabio Giovanazzi

Challenge 4: EncMachinery™

Our EncMachinery™ algorithm is so secure that we even let you look at encrypted company secrets…

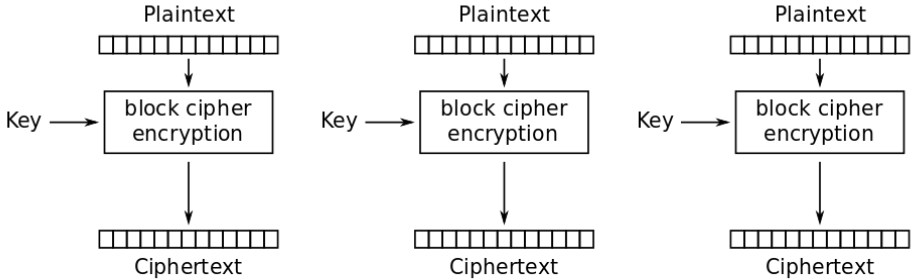
Hints:

- Take a look at the **ECB** block cipher mode
- Each AES block encodes only **two characters** from the original message at a time

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

- The AES cipher is only able to encrypt 16 bytes at a time
- Therefore the message needs to be split in blocks

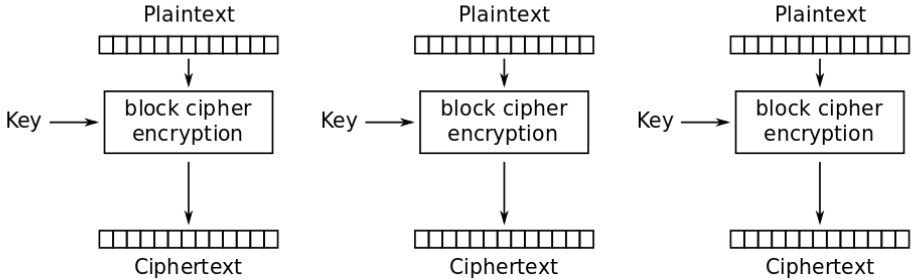


Electronic Codebook (ECB) mode encryption

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

- The AES cipher is only able to encrypt 16 bytes at a time
- Therefore the message needs to be split in blocks
- The **ECB block cipher mode** encrypts each block separately
- So if two blocks have the same plaintext, they will also generate the **same ciphertext**



Electronic Codebook (ECB) mode encryption

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

We can notice that the `encrypt()` function hex-encodes the incoming message 3 times before actually encrypting it. This means that every byte ends up using $2^3=8$ bytes, so a pair of bytes uses a full 16-bytes AES block.

```
def encrypt(m: bytes) -> bytes:
    # a bit of encoding to make hackerz more confused
    for _ in range(3):
        m = m.hex().encode("utf-8")
    m = pad(m, BLOCK_SIZE)
    return cipher.encrypt(m)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

- The number of pairs of bytes is just 256*256=65536 (or even less if we only consider printable characters)

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

- The number of pairs of bytes is just 256*256=65536 (or even less if we only consider printable characters)
- We can brute-force the flag by getting from the server the encrypted block corresponding to each possible pair of characters

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

- The number of pairs of bytes is just 256*256=65536 (or even less if we only consider printable characters)
- We can brute-force the flag by getting from the server the encrypted block corresponding to each possible pair of characters
- Then we can take the encrypted flag block by block, and see which character pair each block corresponds to

© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

```python
import pwn
import string

def splitEvery(s: bytes, n: int) -> list:
    """splits a byte array (or a string) into chunks of length n"""
    return [s[i:i+n] for i in range(0, len(s), n)]

#r = pwn.process(["python3", "challenge.py"])
r = pwn.remote("cyberchallenge.disi.unitn.it", 10101)

# receive the encrypted flag from the server
r.recvuntil(b": ")
flag = splitEvery(r.recvline(keepends=False), 32)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

```
# build a long message with all possible pairs of characters
FLAG_CHARS = string.printable
message = ""
for c1 in FLAG_CHARS:
    for c2 in FLAG_CHARS:
        message += c1 + c2

# send to the server our long message and obtain the encrypted blocks
r.sendlineafter(b"? ", message.encode("utf-8").hex().encode())
r.recvuntil(b": ")
encrypted_char_pairs = splitEvery(r.recvline(keepends=False), 32)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 4 solution

```
# decrypt the flag one pair of characters at a time
for flag_char_pair in flag:
    try:
        index = encrypted_char_pairs.index(flag_char_pair)
        print(message[index*2 : index*2+2], end="")
    except ValueError:
        print("??", end="")
print()
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Some history first:

- Why is DES not used anymore?
  Key space is very small: 2^{56}

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Some history first:

- Why is DES not used anymore?
  Key space is very small: 2^{56}

- Is Double DES a solution?

```
double_encrypt(msg) =
    encrypt(key2, encrypt(key1, msg))
```

  Key space: $2^{56 + 56} = 2^{112}$, right?

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Yes, but there is a problem: **Meet In The Middle Attack**

- Suppose we have a message **`msg`** and its encryption **`ct`**. We can brute force the keys with the following idea:

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Yes, but there is a problem: **Meet In The Middle Attack**

- Suppose we have a message **`msg`** and its encryption **`ct`**. We can brute force the keys with the following idea:

- Since

```
ct = encrypt(key2, encrypt(key1, msg))
```

then

```
decrypt(key2, ct) = encrypt(key1, msg)
```


© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

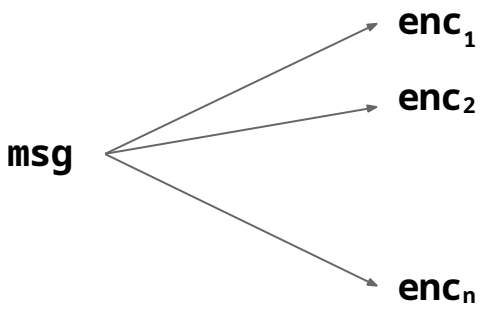
- We want to find a collision between the decryptions of **`ct`** and the encryptions of **`msg`**

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

- We want to find a collision between the decryptions of **`ct`** and the encryptions of **`msg`**

Compute the *encryption* of **`msg`**
with all possible **`key1`**

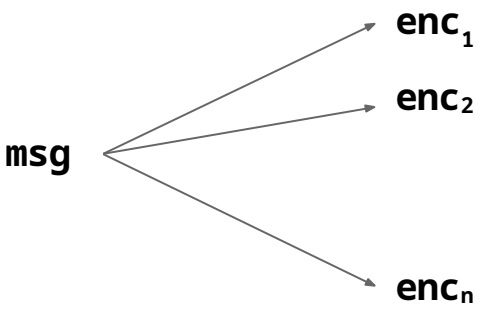


© Carmen Casulli, Fabio Giovanazzi

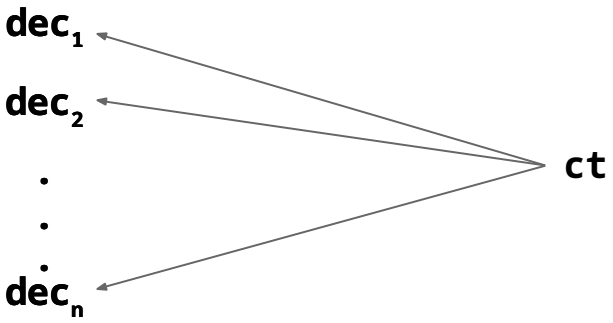
Challenge 5: DoubleDes

- We want to find a collision between the decryptions of **`ct`** and the encryptions of **`msg`**

Compute the *encryption* of **`msg`** with all possible **`key1`**



Compute the *decryption* of **`ct`** with all possible **`key2`**



© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

- If $\mathbf{dec}_i = \mathbf{enc}_j$ then we must have that
  $\mathbf{key1} = \mathbf{key}_j$ and $\mathbf{key2} = \mathbf{key}_i$

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

- If $\mathbf{dec}_i = \mathbf{enc}_j$ then we must have that
  $\mathbf{key1} = \mathbf{key}_j$ and $\mathbf{key2} = \mathbf{key}_i$
- Total cost of the attack:
  2^{56} of space
  $2 * 2^{56} = 2^{57}$ encryptions

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Now it’s your turn!

© Carmen Casulli, Fabio Giovanazzi

Challenge 5: DoubleDes

Now it’s your turn!

Hints:

- The first block of plaintext is known: `"The flag"`

© Carmen Casulli, Fabio Giovanazzi

Challenge 6: Encrypted cookies

Can you *check* if my session cookie *checks* are correct?

```
try:
    cookie = input("Give me a session cookie to check (in hex)? ")
    cookie = bytes.fromhex(cookie.strip())
    iv, encrypted = cookie[:BLOCK_SIZE], cookie[BLOCK_SIZE:]

    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = cipher.decrypt(encrypted)
    decrypted = unpad(decrypted, BLOCK_SIZE)
    print("TODO implement actual validity check")

except ValueError:
    print("Invalid session cookie")
```

Challenge 6: Encrypted cookies

Can you *check* if my session cookie *checks* are correct?

Hints:

- We get two different responses based on whether the **padding** is correct or not, since `unpad()` throws `ValueError` if the padding is wrong. This is an **oracle**!

© Carmen Casulli, Fabio Giovanazzi

Challenge 6: Encrypted cookies

Can you *check* if my session cookie *checks* are correct?

Hints:

- We get two different responses based on whether the **padding** is correct or not, since `unpad()` throws `ValueError` if the padding is wrong. This is an **oracle**!
- CBC is **malleable**

© Carmen Casulli, Fabio Giovanazzi

Challenge 6: Encrypted cookies

Can you *check* if my session cookie *checks* are correct?

Hints:

- We get two different responses based on whether the **padding** is correct or not, since `unpad()` throws `ValueError` if the padding is wrong. This is an **oracle**!
- CBC is **malleable**
- The padding scheme is the default one for `pad()`: `'pkcs7'`

© Carmen Casulli, Fabio Giovanazzi

Challenge 6: Encrypted cookies

Can you *check* if my session cookie *checks* are correct?

Hints:

- We get two different responses based on whether the **padding** is correct or not, since `unpad()` throws `ValueError` if the padding is wrong. This is an **oracle**!
- CBC is **malleable**
- The padding scheme is the default one for `pad()`: `'pkcs7'`
- The last block already ends with correct padding, which may break your logic, so be careful

Challenge 6 solution

- We can perform a **padding oracle attack** on **CBC block cipher mode**
- Refer to the lessons slides and to the Python solution for an explanation

© Carmen Casulli, Fabio Giovanazzi