EXERCISE 1

Consider the code of the oracle *server1.py* (see the material).

Decrypt the message.

ciphertext =
b';s\xf5\x8d\xd0\x8d\xeb\x02&A\xa4\xff\xd2\xc3%ht\x95\xe1)\xb2\xc
c\xfc\xee\x18\x12}\x11)\xee\xec\xa6'

EXERCISE 2

Consider the code of the oracle *server2.py* (see the material).

Modify the *guess_byte()* function (in *client2.py*) to decrypt all but the last block of this ciphertext

cbc_oracle_ciphertext = b"\x94\x99d{SE,\x00I|\x04\xfe\x14N5\xb4\x1c\xadk\x9f\xfc\xceg8'\xe6{\x13\xc9\x94\xce\xae\x0b\x97\xfbm\x8b0\n\\\x94-~\xa7CH\xbf\x150\x88:(\x8c\xeb\xbf5\xaf\xdd_5\xe9\xb4f\xf9"

HOMEWORK: complete the modifications to decrypt the whole message.

Exercise 3

A smart cryptographer was asked to generate an RSA key for generating digital signatures.

To optimize the performance of the server only generating signatures, he inverted the public and private exponents (e=the long number, d=F4), as they are commutative and F4 has only two bits in its binary representation.

Even assuming that attackers cannot guess the private exponent, what attacks can be mounted?

What is the additional side effect of this smart approach?

Exercise 4

You have sniffed 100 digests of non-salted passwords encrypted with Salsa20 and a fixed nonce

Plan an attack based on the code here

https://github.com/aldobas/cryptography-03lpyov-exercises/blob/master/AY2122/Python/attacks/KeyStreamReuse/keystream_reuse_attack.py

- what version of the attack would you try?
- what modifications did you plan?

Exercise 5 (at home)

Manually implement the length extension attack with the SHA1 implementation provided in the *sha1.py* file.

You have all the data there