Part B. Coding/Theoretical Problems (8 marks)

Part B.1. Coding Problem: Padding Oracle Attack

Consider the following encryption scheme:

- $\mathsf{Enc}(k, m)$: The message m is an arbitrary sequence of bytes. Here the key k is a 16 byte string. We use AES in CBC mode to encrypt this message. Since AES can only handle messages whose length (in bits) is a multiple of 128, we have to pad m appropriately.

  Padding Scheme: Instead of padding at the right-most end, we would instead pad at the left-most end (as suggested by one of the students in class). Let p be the number of bytes to be padded – then include the number p (in binary) in each of the p bytes.

  Examples:

  - m = (11 42 33 01 89 12). This message is 6 bytes long. We need to pad it with 10 bytes. The resulting message m' would be

    $$m' = (10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 11\ 42\ 33\ 01\ 89\ 12).$$

  - m = (02 02 00 00 00 00 00 00 00 00 00 00 00 00 00 16). This message is 16 bytes long. Add a new padding block to avoid ambiguity. The padded message

    $$m' = (16\ 16\ \ldots 16\ 16\ 02\ 02\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 16).$$

- $\mathsf{Dec}(\mathsf{ct}, k)$: Decryption algorithm simply decrypts the ciphertext to obtain the padded message $m' = (y_1, y_2, ..., y_\ell)$ where each y_i is 16 bytes long. It checks if y_1 is a valid padded string. That is, check that the first byte of y_1 is a number between 1 and 16. If the number is z, then check that the next $z - 1$ bytes after it have the value z. If any of these is violated, output "Error: Bad Padding". Otherwise, output the decrypted string (without the padding).

Files Given: Since the messages and the cipher-texts are arbitrary sequences of bytes, we represent each of them by a list of integers within the range [0, 255]. You are given the following `python` files on MS Teams (`A2_Coding_Students.zip`):

- `encrypt.py`:
  - This file has a 16-bit key hardcoded into it for the AES scheme
  - It has a function called `encrypt(message)`, which takes a list of integers as input, pads it appropriately and returns the encrypted bytes
  - This script can be used to generate cipher-texts with the given key. You can use it to check the correctness of your code.
- `decrypt.py`:
  - This file has the same key hardcoded for AES as in `encrypt.py`

  - It has a function called `check_padding(encd)`, which takes a ciphertext as input (in the form of an integer list), decrypts it and checks for a valid padding.
  - It returns 0 if it was a valid padding, else returns 2. It does NOT return the decrypted message

- `attack.py`:
  - You are required to implement your attack in this file. The function to be implemented is `attack(cipher_text)`
  - It is supposed to take a ciphertext as input, and the return the original message (as a list of integers)
  - You are allowed to make calls to `check_padding()` from `decrypt.py`

Instructions:

- You are only required to submit `attack.py`, with your implementation of `attack()`. You don't need to submit `encrypt.py`, `decrypt.py`.
- Submit the `attack.py` file on Gradescope in the `Assignment2 Coding4`
- Your submission will be checked on ciphertexts which are encryptions of some messages with some key k
- During grading, we would change the key so you cannot simply decrypt the ciphertext