6. (5 marks) **CBC mode with bad initialization**

Problem Description: CBC mode is a commonly used provably secure encryption scheme. While the encryption scheme is in itself simple, it is still possible for things to go horribly wrong in practice when in order to make the implementation of the scheme simpler, implementations end up using "bad" initialization vector (IV), such as directly picking up the last block of the previous ciphertext or even the key itself. In this problem, we will explore an attack on the latter implementation. Hopefully, this shows that something as simple as choosing the IV can turn out to be fatal if done incorrectly and that a random IV is the correct way to use CBC mode of encryption.

For the purpose of this question, consider the IV supplied at the start to be same as the key used with the block cipher(AES).

You are given an encryption of some message, and you are supposed to recover the secret key by making only one query to the decryption oracle.

How this attack is applicable in the real world — consider the following scenario: there is a server that communicates with clients using a shared secret key. Whenever the server receives a 'bogus' ciphertext (that is, a ciphertext that results in some nonsensical decryption), it outputs an error message containing the decrypted message (this happens quite often in practice when code for debugging is pushed into production). If the server receives a proper ciphertext, it computes the next message and outputs an encryption of the next message.

Therefore, the attacker has access to ciphertexts, and also has access to a partial 'decryption oracle'. In this assignment, we are giving you access to the full decryption oracle.

Files Given: You are given the following Python files on Teams (`COL759_A2_Coding3.zip`)

- `decrypt.py`:
  - This file has a `decryption_oracle` class which has 16-byte key for the AES scheme hard-coded (the keys will be changed during autograding)
  - It has a function called `decrypt(ct)`, which takes a byte array of size 48 and returns the decrypted message m of 48 bytes in the byte format.
    An example of how the message and the ciphertext (format of both) will look like is given below(not indicative of actual size used in this):
    m =`b'd2\xad\xbc\x8d\xc1R\xcc\xa3\x8c\x9d\x10\x8d\xfc'`
    $ct = [137, 110, 41, 18, 99, 62, 3, 4, 197, 186, 163, 215, 12, 48, 176, 44]$
  - This `decrypt` function can be used only once for your attack, If you use it for more than once then it will just return None.
- `attack.py`:
  - You are required to implement the `attack(ciphertext, decrypt)` function in this file.
  - You are given a cipher on some message of 48 bytes length(fixed).
  - It is supposed to return the key which is used during the encryption and decryption of the scheme.

  - You have to return the key in byte format only.
  - You are allowed to make calls to `decrypt(ct)` function of `decrypt.py`.

Instructions:

- You are only required to submit `attack.py` on Gradescope in the `Assignment2 Coding3` with your implementation of `attack(ciphertext, decrypt)`. You don't need to submit any other files. All test cases are public, you should be able to find the number of test cases that your code passes on Gradescope.
- Provide a high-level description of your attack in the pdf submission.