CSCI 3330 Assignment 6

Deadline: November 6, 2025, 10:00pm

1 Background

In this assignment, you will be developing and executing a kind of chosen ciphertext attack known as a CBC padding oracle attack. Specifically, by completing this assignment, you will:

- Implement an `oracle()` that distinguishes valid vs. invalid PKCS#7 padding from server responses.
- Exploit CBC malleability to adaptively manipulate ciphertext, induce target padding patterns, interpret responses to infer plaintext bytes, and resolve ambiguous cases.
- Decrypt the plaintext of a multi-block AES-CBC-encrypted message without knowledge of the key.

2 AI policy

You may use AI to learn background concepts. You may not use AI to generate, verify, or iterate on solutions for this assignment.

Allowed:

- Concept definitions and explanations: "What is a CBC padding oracle?", "What is CBC mode?", "What is AES?"
- Generic examples not derived from this assignment: "Walk through an example of how a CBC padding oracle attack works at a high level," "Walk through an example of how to discover a CBC padding oracle."
- Non-generative tooling: man pages, official documentation, calculators, programming languages, and trying something for yourself.

Handle with care:

- Tools which improve grammar or clarity.
- IDE autocompletion or documentation lookup that does not inject solution content.

Not allowed:

- Any hint, outline, or solution for these specific questions (including rephrased or "similar" versions), e.g., "Help me with the attack in Q2," "Write a CBC padding oracle attack," or "Check my answers."
- Pasting assignment text or numbers into AI to tailor examples.
- Using AI to generate any text, code, calculations, or any other artifacts that you submit.

Ask me questions instead! When in doubt, don't use AI, because **you won't have access to it on tests**, for which these assignments are your preparation.

3 Attack

In this assignment, you will implement `crack.py`, a CBC padding oracle attack to decrypt an encrypted message for which you do not have the key (only an RSA-encrypted copy of the key). You will be working with the files in the tarball here to attack the following encrypted request:

```
https://brainhammer.com/cbc/?c=58d9f18eb1a68a928dc14ad17527f6bc04
ddf9ca4aac64fb0aa83cdd5ab54c4185e8870321d3b24c78eb21057a961dad&iv
=e486b3afb05e477d1ec8e2dae6efdc74&k=51905de251e6b496307787fd81046
23a0bcdf895377ad56b42836c19db39f4b64f0f15008487a98db6e4183cc9ab00
1a8e3d09d1bb44c3553f8ccb27a76302279a4d02477eeea0a61c82039feefb0b8
5736c8de2a108066e483cafce7f2dff77a6958090a200355f786b45c5a92c3074
c8f1a926bee2b6a85cb9c83d1c643116c971c429c3aaafcd0698f5f85b8097f8d
157d492ec9bada481822398abf530e38b4ff7eb0248bb4cb6d3a6204a02236ffb
28cb50af6c3ec63c7c880635431fe6e77855c2246a6b66c0339e78c97854fae55
0de47f2bf6b90076ce2d5e6aa1758d78afc9ff7c60ab88c8c727b2756c3090a6c
b47c88bdb57d2c44b73d5b399239
```

At at high level, `c` corresponds to the encrypted payload, `iv` corresponds to the IV with which it was encrypted, and `k` corresponds to the RSA-encrypted key with which the payload was AES-encrypted in CBC mode (all of these values are hex-encoded). To understand the protocol at a deeper level, consult `client.py`, a sample implementation which encrypts and transmits a sample message (but one that is malformed since we don't know the intended format of the message yet). The linked tarball also includes `crack-template.py`, which you may use as a template for implementing `crack.py`. Note that you will need to install the `cryptography` python package to run `client.py` or `crack.py`.

When implementing the code for this assignment, you may wish to familiarize yourself with python's `bytearray()` versus `bytes()` objects. Specifically, the former is mutable, whereas the latter is immutable. Part of the challenge in organizing your code for this assignment will be in knowing when to mutate blocks versus make copies of them.

1. Implement the `oracle()` function. This function takes a payload, IV, and encrypted key, and, by analyzing the server's response, returns `True` if the padding is correct, else `False`. **Hint:** to determine what response signals incorrect padding, consider modifying the message you are attacking in `client.py` to result in incorrect padding when decrypting.

2. Implement the `update_padding_suffix()` function. In a CBC padding oracle attack, you decrypt a block by carefully making modifications to the block previous to it and observing how doing so changes the output of the server. This function takes a "previous block" `bytearray` known to induce, upon decryption of the "current block", a correct padding of (`pad_len`-1) and returns a block which will induce the padding byte `pad_len` in the final (`pad_len`-1) bytes of the decrypted "current block". Modifications should be done in-place on the input `bytearray`. Example test vectors:

```
>>> block = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
>>> update_padding_suffix(block, 1)
>>> block
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

>>> block = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
>>> update_padding_suffix(block, 2)
>>> block
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

>>> block = [0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3]
>>> update_padding_suffix(block, 4)
>>> block
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4]

>>> block = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
>>> update_padding_suffix(block, 4)
>>> block
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 9, 8, 23]
```

3. Implement the `run_padding_probe()` function. This function takes the "previous block" you are modifying, the "current block" you are trying to decrypt, an IV, an encrypted key, and the padding length that you are attempting to induce on the current block when decrypted. You should test all 256 possible values of byte at index (`BLOCK_SIZE` - `pad_len`) of the "previous block" to determine which of these induce correct padding when the "current block" is decrypted. Its return value is a list of each such block inducing correct padding.

4. Implement the `resolve_last_byte_tie()` function. When `pad_len` = 1, there can be two possible byte values that you find induce correct padding. Why? This function takes as arguments a list of candidate blocks for

inducing a correct padding of `pad_len`, the "current block" that you are trying to decrypt, an IV, and an encrypted key. In this function you should disambiguate the case when `pad_len` = 1 by, for each candidate block, modifying its second-to-last byte and re-querying `oracle()`. Return from this function the block that induces a correct padding of `pad_len`.

5. Implement the `decrypt_block()` function. This function takes as arguments the "previous block" which you will be modifying, the "current block" which you will be decrypting, an IV, and an encrypted key. Using this function you should call `run_padding_probe()` to decrypt each byte of the "current block" beginning with the last and moving leftward. You must handle the ambiguous case when `run_padding_probe()` returns more than one candidate by calling `resolve_last_byte_tie()`.

6. Implement the `decrypt()` function. This function takes as arguments an encrypted payload, which you will be decrypting, the IV with which it was encrypted, and the key (RSA-encrypted) with which it was AES-encrypted in CBC mode. This function should return the decrypted payload.

Use `decrypt()` to decrypt the encrypted payload. What is the decrypted payload?

4 Submission

Email your code and the decrypted payload in a single `*.txt` or `*.pdf` file attachment to my email address in the Syllabus.