ICT2213—Applied Cryptography

Puzzle #2: CBC padding oracle attack

1 Introduction

The *padding oracle attack* in CBC mode relies on the fact that some systems could return an error when the attempted decryption of a ciphertext results in an incorrect padding. For example, a web server could return a response code of 500 when processing a request that involves the incorrect decryption of a ciphertext. Recall that, under PKCS7 padding, the last incomplete block must be padded as follows (assuming a 16-byte block size):

- 15-byte block: add byte `0x01`
- 14-byte block: add bytes `0x02 0x02`
- 13-byte block: add bytes `0x03 0x03 0x03`
- etc.

Under this attack, you can repeatedly send a carefully crafted ciphertext to the server, and gradually brute-force (one byte at a time) an existing ciphertext that you have intercepted, based on the server's responses (on whether the padding was correct or not). Before continuing, you should first understand the attack. There are a lot of Internet resources you can find that describe the CBC padding oracle attack in detail (web pages and YouTube videos). Once you are familiar with the attack, you will understand that it is performed as follows:

1. The attacker has intercepted a CBC-encrypted ciphertext. Assume that the ciphertext consists of n individual blocks, $C_1, C_2, \ldots, C_n$ (and the plaintext IV).
2. For each ciphertext block C_i, the attacker will construct a ciphertext $R||C_i$ and will send it to the remote server for decryption (R will be the first block of the ciphertext and C_i will be the second). R is a block that is crafted by the attacker, and will change every time the attacker interacts with the server (but C_i remains unchanged). The IV in this case is not important and can be any random value. With every interaction, the server will respond whether the padding was correct or not. Based on the response, the attacker will modify R accordingly and will continue to do that until the entire (intermediate) plaintext P'_i is revealed. Note that this is not the final plaintext, since it has to be XORed with C_{i-1} to recover P_i. Similarly, P'_1 must be XORed with the original IV that is included with the intercepted ciphertext.
3. The attacker repeats this process for all C_i and then outputs the plaintext $P_1, P_2, \ldots, P_n$.

2 Your task

You must write a Python script that accepts two command-line arguments. The first one is a file that contains the base64-encoded IV, and the second one is a file that contains the base64-encoded ciphertext. The script should print the entire plaintext. A sample command to execute your code looks as follows:

```
$ python3 cbc.py iv.txt ciphertext.txt
```

Since we do not have a server to run the attack, we will use an oracle to respond to your decryption queries. Specifically, for testing your code, I will use the following script, named `oracle.py`. The value of the IV at the oracle is irrelevant, so any value will do. However, when testing your code, I will change the value of the key to match the key that encrypted the original ciphertext. The output of the oracle will be either *Yes* to indicate a correct padding or *No* to indicate a padding error.

```
import sys, base64
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding

if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} ciphertext\n")
        sys.exit()

ct = base64.b64decode(sys.argv[1])

iv = b'AAAAAAAAAAAAAAAA'
key = b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
decryptor = cipher.decryptor()
data = decryptor.update(ct) + decryptor.finalize()

unpadder = padding.PKCS7(128).unpadder()
pt = unpadder.update(data)
try:
    fin = unpadder.finalize()
except ValueError:
    print("No")
    sys.exit()

print("Yes")
```

Make sure your code calls the oracle as follows (as an example) to test different ciphertexts:

```
$ python3 oracle.py U8q4u814+gUFZq53zfvCKtatdImn0Qkv4qmceANoIek=
```

To test the correctness of your code, select a key and IV to create the original ciphertext, and store the IV and ciphertext in separate files (in base64 encoding). Next, modify the `oracle.py` program to hardcode your own key, and then run the attack. If the attack is successful, you should be able to see the plaintext.