Problem 3

3.1. Approach to Recovering the Cookie

1. **Finding the Cookie Length**
   - Send requests with incrementing path lengths and observe the corresponding ciphertext length.
   - When the ciphertext size jumps by one full block (16 bytes), it indicates that padding has caused a new block to be added.
   - Using this observation, I derived the exact cookie length by subtracting the known prefix (";cookie=") and the padding adjustment.
2. **Byte-by-Byte Recovery via CBC Chaining.**
   - **Step 1: Align the Unknown Byte.**
     We choose a path length so that the i-th cookie byte appears at the final position of an AES block.
   - **Step 2: Obtain a "Reference" Block for the Next Encryption.**
     We send a "setup" request (device(prefix)) to get a ciphertext whose final block becomes the IV for the next request.
   - **Step 3: Brute Force Each Cookie Byte (0–255).**

     Constructs the input, specifically input=（known_part+guess）XOR C_i XOR C_last. If the guess is correct, the corresponding block in the ciphertext obtained by calling the device(input) at this point is the same as the block we got from the previous block.

     C_i is the block that contains the i-th byte of the cookie. C_last is the last block.
   - **Step 4: Repeat for All Cookie Bytes.**

3.2. Runtime Analysis

Let n be the length of the cookie.

- For each byte of the cookie, we generally perform:
  a. **One setup call** .
  b. **Up to 256 guess calls** (worst-case) to test each possible byte value.

Hence, the total number of oracle calls is approximately $n \times (1 + 256) = 257n$. In **big-O** terms, it is $O(n)$, more precisely $O(256n)$.

Problem 4

4.1. Insecure Use of CBC-MAC

- **CBC-MAC Definition (simplified).**
  For blocks $M_1, M_2, \dots, M_\ell$ of a message M, with an all-zero IV, the CBC-MAC is T_ℓ, where

$$T_i = E_k(T_{i-1} \oplus M_i), T_0 = 0.$$

- **Vulnerability to Variable-Length Messages.**
  If the receiver does not "bind" the length of M into the MAC (e.g., by prepending or appending the length), an attacker can perform a *chaining trick* to link partial computations.

4.2. Attack Demonstration

1. **Query the MAC of message A.**
   Let T_A be the MAC of A.

2. **Query the MAC of message B.**
   Let T_B be the MAC of B.

3. **Construct a New Message M^*:**

$$M^* = A \parallel (T_A \oplus B_1) \parallel B_2 \parallel \dots \parallel B_\ell,$$

where $B_1, B_2, \dots, B_\ell$ are the blocks of B.

   - After processing A, the internal state is T_A.
   - By adding a "bridging block" $T_A \oplus B_1$, the CBC decryption at that stage yields B_1 as if we are continuing with message B.
   - Consequently, the final CBC-MAC block computed for M^* will match T_B.

Therefore, $M^* \neq A$ and $M^* \neq B$, yet you have produced a valid MAC for M^*. This is a forgery demonstrating why CBC-MAC is insecure for variable-length messages under a single key without length binding.