Homework 4

Task 1 - Encrypt & MAC (15 points)

(a)

From the ciphertexts, we can infer that the plaintexts are the same.

Justification:
We can see that the last 256 bits of both ciphertexts C_1 and C_2 are the same. Given that the tag produced by the MAC algorithm is also 256 bits, we can infer that these common bits are the MAC tags.

Since the MACs are done on the plaintexts and both ciphertexts are generated using the same secret key, given that the MAC tags are equal, this implies that the actual plaintexts are the same.

Additionally, since both ciphertexts are encrypted using CTR with AES, the encryption algorithm will produce different ciphertexts even if the plaintexts are the same (assuming C_1 and C_2 used different initialization vector). Thus, the fact that the first 50 bytes of C_1 and C_2 are different doesn't contradict our inference that the plaintexts are the same.

(b)

E&M is a good scheme in terms of IND-CPA security.

Explanation:
Since HMAC with SHA-256 produces tags which are computationally indistinguishable from random by someone without the key, the tags themselves do not reveal any information about the plaintext. Since CTR mode with AES is IND-CPA secure, the ciphertext produced does not allow an attacker to determine which plaintext was encrypted.

Appending a MAC tag to the ciphertext is similar to appending a fixed pattern to an IND-CPA secure ciphertext, which we have proven does not impact the scheme's IND-CPA security in Homework 2 Task 4b). The tag acts similar to a fixed suffix that provides integrity without compromising confidentiality from the CTR encryption.

Therefore, the E&M is a good scheme in terms of IND-CPA security.

(c)

The main observation is that the E&M scheme produces identical tags for the same plaintexts since SHA-256 is deterministic. Let $\Pi = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ be the encryption scheme E&M, We can construct an adversary against Π in the following way:

$$\begin{array}{l} \underline{\textbf{adversary } A^{O}():} \\ C_1'[0]C_1'[1]...C_1'[\ell] \mid\mid T_1 \leftarrow O.\mathsf{Encrypt}(0^{n\cdot\ell}) \\ C_2'[0]C_2'[1]...C_2'[2\ell] \mid\mid T_2 \leftarrow O.\mathsf{Encrypt}(0^{n\cdot 2\ell}) \\ C^* = C_2'[0]C_2'[1]...C_2'[\ell] \mid\mid T_1 \\ \textbf{return } C^* \end{array}$$

Consider the INT-CTXT oracle. When running $\mathsf{A}^{\mathsf{INT\text{-}CTXT}[E\&M]}$, for the adversary to succeed, two things need to be satisfied: (1) $C^* \notin \mathcal{Q}$, and (2) $\mathsf{Dec}(K, C^*) \neq \perp$.

Firstly, the oracle's state is $\mathcal{Q} = \{C_1, C_2\}$, where $C_1 = C_1'[0]C_1'[1]...C_1'[\ell] \mid\mid T_1$ from $\mathsf{Enc}(K, 0^{n\cdot\ell})$, and $C_2 = C_2'[0]C_2'[1]...C_2'[2\ell] \mid\mid T_2$ from $\mathsf{Enc}(K, 0^{n\cdot 2\ell})$. Assuming the IV is different for each encryption, $C_2'[0] \neq C_1'[0]$ and because C^* is composed only half of the blocks from C_2 with the tag T_1 from C_1, we have that $C_1 \neq C_2 \neq C^*$, and thus, $C^* \notin \mathcal{Q}$.

Secondly, by the E&M construction, decrypting the counter-mode ciphertexts of C_1 and C^* will yield the same plaintext, because the first ℓ blocks of $0^{n\cdot 2\ell}$ are the same as $0^{n\cdot\ell}$, and CTR blocks can be decrypted independently. Since SHA-256 is deterministic, running the HMAC algorithm on the decrypted counter-mode ciphertext of C^* will produce the same tag as T_1. Therefore, $\mathsf{Dec}(K, C^*)$ returns $M \neq \perp$.
In conclusion,

$$\begin{aligned} \mathsf{Adv}_{\Pi}^{\text{int-ctxt}}(A) &= \Pr[\mathsf{A}^{\mathsf{INT\text{-}CTXT}[\Pi]} \Rightarrow 1] \\ &= \Pr[C^* \notin \mathcal{Q} \wedge \mathsf{Dec}(K, C^*) \neq \perp] = 1, \end{aligned}$$

and since A returns in polynomial time, this means Π is not INT-CTXT secure.

Task 2 - Authenticated Encryption (15 points)

(a)

The scheme doesn't guarantee ciphertext integrity.
Explanation:
The main observation is that the encryption scheme ensures the integrity of the decrypted plaintext by XORing all plaintext blocks. Since counter-mode encryption is malleable, two bit-flips at the same index in different blocks will cancel each other out. This allows us to construct ciphertexts that will pass the integrity check but differ from the original ciphertext. Let $\Pi = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ be the AES-based CTR encryption scheme described in the question. We can construct an adversary against Π as follows:

adversary $A^{O}()$:
$C[0]||C[1]||C[2]||C[3]||C[4] \leftarrow O.\mathsf{Encrypt}(0^{256})$
$C^*[1] \leftarrow C[1]$ with the first bit flipped
$C^*[2] \leftarrow C[2]$ with the first bit flipped
$C^* = C[0]||C^*[1]||C^*[2]||C[3]||C[4]$
return C^*

Consider the INT-CTXT oracle. When running $\mathsf{A}^{\mathsf{INT\text{-}CTXT}[\Pi]}$, for the adversary to succeed, two conditions must be met: (1) $C^* \notin \mathcal{Q}$, and (2) $\mathsf{Dec}(K, C^*) \neq \perp$.

Firstly, the oracle's state is $\mathcal{Q} = \{C\}$, where $C = C[0]||C[1]||C[2]||C[3]||C[4]$ from $\mathsf{Enc}(K, 0^{256})$. Given the 256-bit plaintext and 128-bit block size, we know that $C[0]$ is the initialization vector, $C[1]$ and $C[2]$ are the encrypted plaintext blocks, $C[3]$ is the padding block, and $C[4]$ is the additional integrity checking block. As C^* has the first bits of $C[1]$ and $C[2]$ flipped, it is clear that $C \neq C^*$, and thus, $C^* \notin \mathcal{Q}$.

Secondly, by the construction of the encryption scheme, $\mathsf{Dec}(K, C^*)$ recovers $M^*[1], M^*[2], M[3], M[4]$ from C^* in counter mode. Due to the malleability of counter mode, the first bit of $M^*[1]$ and $M^*[2]$ will be flipped compared to the original plaintext blocks $M[1]$ and $M[2]$. However, since the flips occur at the same position, $M^*[1] \oplus M^*[2] \oplus M[3] = M[1] \oplus M[2] \oplus M[3] = M[4]$. Since the padding block $M[3]$ is unchanged, $M^*[1], M^*[2], M[3]$ will pass the padding validation. Therefore, $\mathsf{Dec}(K, C^*)$ returns $M \neq \perp$.
In conclusion, the adversary's advantage is computed as

$$\begin{aligned}\mathsf{Adv}_{\Pi}^{\text{int-ctxt}}(A) &= \Pr[\mathsf{A}^{\mathsf{INT\text{-}CTXT}[\Pi]} \Rightarrow 1] \\ &= \Pr[C^* \notin \mathcal{Q} \wedge \mathsf{Dec}(K, C^*) \neq \perp] = 1,\end{aligned}$$

which indicates that Π is not INT-CTXT secure since A operates in polynomial time.

(b)

The modified scheme still doesn't satisfy ciphertext integrity.
Explanation:
The main observation is that the modified scheme remains vulnerable to malleability under counter mode encryption. Flipping the same bit in two different blocks of the plaintext will not change the result of the XOR operation. Since AES is A deterministic function, an adversary can still produce a modified ciphertext that, when decrypted, still passes the integrity check.
Consider the modified AES-based CTR encryption scheme $\Pi' = (\text{Kg'}, \text{Enc'}, \text{Dec'})$ with an added integrity check, where the last ciphertext block is computed as $C[\ell+1] = \text{AES}_K(\text{IV} \oplus M[1] \oplus \cdots \oplus M[\ell])$. We can reuse the adversary from part (a) against the integrity of Π' as follows:

adversary $A^{O'}()$:
$C[0]||C[1]||C[2]||C[3]||C[4] \leftarrow O'.\text{Encrypt}(0^{256})$
$C^*[1] \leftarrow C[1]$ with the first bit flipped
$C^*[2] \leftarrow C[2]$ with the first bit flipped
$C^* = C[0]||C^*[1]||C^*[2]||C[3]||C[4]$
return C^*

Consider the INT-CTXT security definition. For adversary A to succeed against the integrity of Π', two conditions must be satisfied: (1) $C^* \notin \mathcal{Q}$, and (2) $\text{Dec'}(K, C^*) \neq \perp$.

Firstly, same as in part(a), since C^* has the first bits of $C[1]$ and $C[2]$ flipped, we have $C \neq C^*$, and thus, $C^* \notin \mathcal{Q}$.

Secondly, $\text{Dec'}(K, C^*)$ first recovers $M^*[1], M^*[2], M[3]$ from C^*. Due to the malleability of counter mode, the first bit of $M^*[1]$ and $M^*[2]$ are flipped from the original plaintext blocks $M[1]$ and $M[2]$. However, the bit flips in $M^*[1]$ and $M^*[2]$ cancel out in the XOR operation used for integrity check. Thus,

$$\text{AES}_K(\text{IV} \oplus M^*[1] \oplus M^*[2] \oplus M[3]) = \text{AES}_K(\text{IV} \oplus M[1] \oplus M[2] \oplus M[3]) = C[4]$$

Since the padding block $M[3]$ remains unchanged, $M^*[1], M^*[2], M[3]$ will pass the padding and integrity checks. Therefore, $\text{Dec'}(K, C^*)$ returns $M \neq \perp$.
In conclusion, the advantage of the adversary A in breaking the integrity of Π' is:

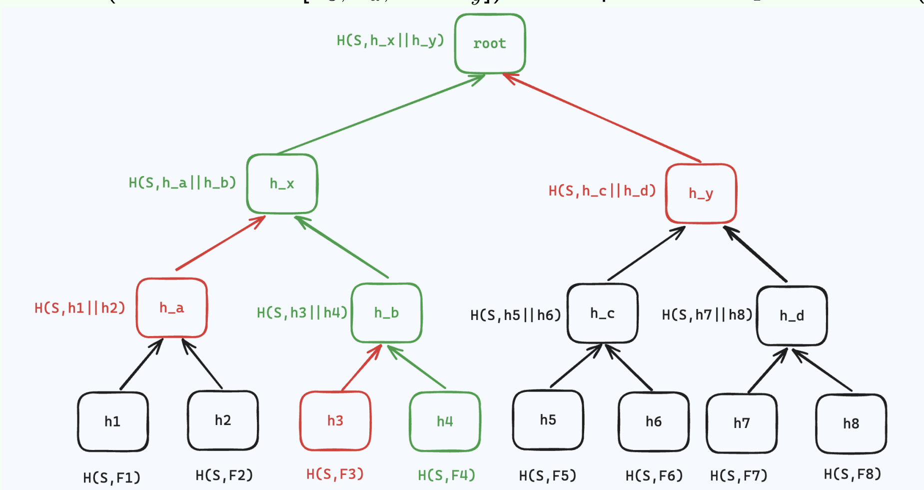
$$\begin{aligned}\text{Adv}_{\Pi'}^{\text{int-ctxt}}(A) &= \Pr[\text{A}^{\text{INT-CTXT}[\Pi']} \Rightarrow 1] \\ &= \Pr[C^* \notin \mathcal{Q} \wedge \text{Dec'}(K, C^*) \neq \perp] = 1,\end{aligned}$$

and since A operates in polynomial time, this implies that Π' is not INT-CTXT secure under this modified scheme.

Task 3 - Set Commitments (10 points)

(a)

To verify the integrity of the file F_i with respect to com, the server needs to provide the client with: (1) The hash value of F_i, $H(S, F_i)$, (2) The sibling hash values for each level of the tree on the path from F_i to the root.
In the example below with $m = 8$, the client wants to retrieve and verify the integrity of F_4. The server sends client the file F_4, the com (the root), and the sibling hash values for each level of the tree (marked in red, $[h_3, h_a$, and $h_y]$) on the path from F_4 to the root (marked in green).



The client can verify the integrity of F_i by doing the following:
1. Compute $H(S, F_i)$ to get the hash of the file.
2. At each level of the binary tree, the client computes the parent node hash by concatenating the hash of F_i and its sibling with the sibling's hash provided by the server, and then hashing the result with the seed S.
3. The client repeats this process up the tree, each time use the newly computed hash along with the next sibling hash provided by the server.
4. When the client reaches the root, they compare the computed root hash with the commitment com published by the server. If they match, then file F_i is valid.

The content sent by the server (excluding F_i itself) is $O(n \log m)$ bits long because each hash is n bits long and there are $\log m$ sibling hashes to send (for a balanced binary tree with m leaves, the path from any leaf to the root has $\log m$ levels [since m is a power of two], and we have one sibling on each level).
Because the hash function H is collision-resistant, it is computationally infeasible for an adversary to find two different inputs that hash to the same output. So changing any file F_i would result in a different hash at the leaf, and change would propagate up the tree result in a different root hash. Therefore, any change in the files would be detected when the client computes a root hash that does not match the commitment.

Task 4 - Number Theory & Groups (10 points)

(a)

By the definition, $\mathbb{Z}_{35}^* = \{a \in \mathbb{Z}_{35} : \text{gcd}(a, 35) = 1\}$. Essentially, we are looking for all numbers between 0 and 34 that are coprime with 35. Since 35 is the product of two prime numbers 5 and 7, any number that is not a multiple of 5 or 7 is coprime with 35. Therefore, $\mathbb{Z}_{35}^* = \{1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34\}$.

(b)

Since 37 is a prime number, $\mathbb{Z}_{37}^* = \{a : 1 \leq a \leq 36\}$. When $g = 2$, we have the following:

e =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	1	2	4	8	16	32	27	17	34	31	25	13	26	15	30	23	9	18
e =	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
	36	35	33	29	21	5	10	20	3	6	12	24	11	22	7	14	28	19

So, we have: $\langle 2 \rangle = \{2^0 = 1, 2^1, ..., 2^{36-1}\} = \mathbb{Z}_{37}^*$.
Therefore, 2 is a generator of $\mathbb{Z}_{37}^*$.

(c)

To find $x \in \mathbb{Z}_{187}$ such that $125x \equiv 4 \pmod{187}$, we are essentially looking for $x, y \in \mathbb{Z}_{187}$ such that $125x + 187y = 4$. We can use the extended Euclidean algorithm to find x and y:

Euclidean algorithm	Backtrack
$187 = 125 \cdot 1 + 62$	$125 - 62 \cdot 2 = 1$
$125 = 62 \cdot 2 + 1$	$125 - (187 - 125) \cdot 2 = 1$

Here we get $125 - (187 - 125) \cdot 2 = 125 \cdot 3 + 187 \cdot (-2) = 1$. Multiplying both sides by 4, we get $125 \cdot 12 + 187 \cdot (-8) = 4$. Thus, we have $x = 12$ and $y = -8$.
Therefore, $x = 12$ is the solution to $125x \equiv 4 \pmod{187}$.