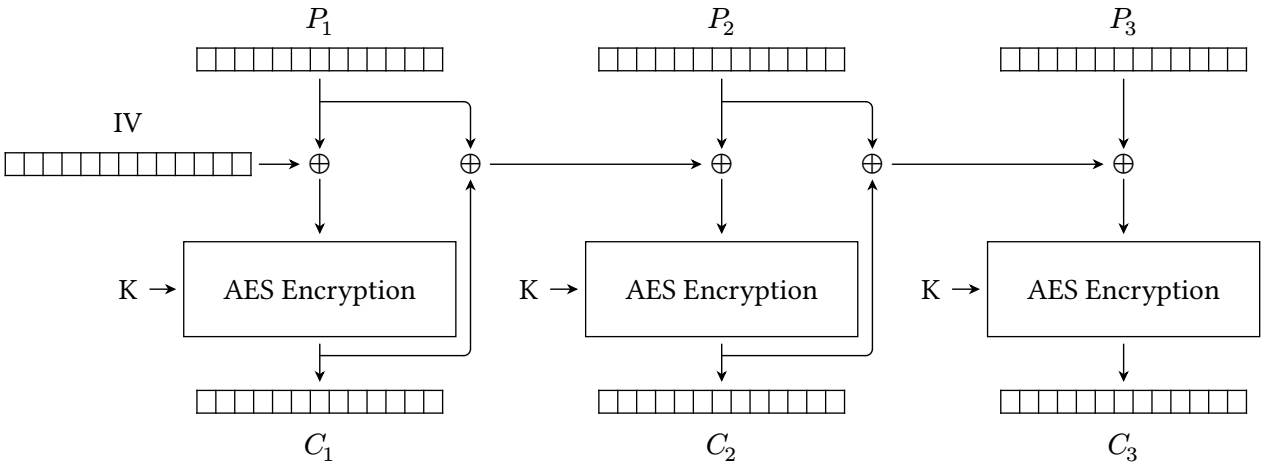
CS 161 Spring 2026 — Introduction to Computer Security — Exam Prep 4

Q1 *EvanBlock Cipher* **(24 points)**

EvanBot invents a new block cipher chaining mode called the EBC (EvanBlock Cipher). The encryption diagram is shown below:



Q1.1 (2 points) Write the encryption formula for C_i, where $i > 1$. You can use E_K and D_K to denote AES encryption and decryption respectively.

$$C_i = E_K(P_i \oplus P_{i-1} \oplus C_{i-1})$$

Solution: For reference, $C_i = E_K(P_1 \oplus \text{IV})$

Q1.2 (2 points) Write the decryption formula for P_i, where $i > 1$. You can use E_K and D_K to denote AES encryption and decryption respectively.

$$P_i = D_K(C_i) \oplus P_{i-1} \oplus C_{i-1}$$

Solution: For reference, $P_i = D_K(C_i) \oplus P_{i-1} \oplus C_{i-1}$

Q1.3 (4 points) Select all true statements about this scheme.

- [x] It is IND-CPA secure if we use a random IV for every encryption.
- [ ] It is IND-CPA secure if we use a hard-coded, constant IV for every encryption.
- [ ] Encryption can be parallelized.
- [ ] Decryption can be parallelized.
- ○ None of the above

Solution: This scheme actually exists in real life; it's called AES-PCBC, where PCBC stands for Propagating Cipher Block Chaining Mode. (The CBC here is the same as the CBC in AES-CBC.)

AES-PCBC is IND-CPA secure with random IVs. Intuitively, notice that AES-PCBC looks quite similar to AES-CBC, except we are sending both the ciphertext and plaintext into the next block cipher encryption, instead of just the ciphertext.

If we use the same IV for every encryption, AES-PCBC is deterministic, so it's not IND-CPA secure.

Encryption cannot be parallelized because you have to wait for the current block's ciphertext to be computed (which requires the current block cipher encryption to run) before you can pass the current block's ciphertext into the next block cipher encryption.

Decryption cannot be parallelized because you have to wait for the current block's plaintext to be computed (which requires the current block cipher decryption to run) before you can pass the current block's plaintext into the XOR that computes the next block's plaintext.

Q1.4 (4 points) Alice has a 4-block message (P_1, P_2, P_3, P_4). She encrypts the message with the scheme and obtains the ciphertext $C = (IV, C_1, C_2, C_3, C_4)$

Mallory tampers with this ciphertext by changing the IV to 0. Bob receives the modified ciphertext $C' = (0, C_1, C_2, C_3, C_4)$.

What message will Bob compute when he decrypts the modified ciphertext C'?

X represents some unpredictable "garbage" output of the AES block cipher.

○ (P_1, P_2, P_3, P_4)	○ (X, X, P_3, P_4)	● (X, X, X, X)
○ (X, P_2, X, P_4)	○ (X, P_2, P_3, P_4)	○ None of the above

Solution: Modifying any ciphertext block in AES-PCBC will cause itself and all future plaintext blocks to become garbage (hence the "propagate").

Alice has a 3-block message (P_1, P_2, P_3). She encrypts this message with the scheme and obtains the ciphertext $C = (IV, C_1, C_2, C_3)$.

Mallory tampers with this ciphertext by swapping two blocks of ciphertext. Bob receives the modified ciphertext $C' = (IV, C_2, C_1, C_3)$.

When bob decrypts the modified ciphertext C', he obtains some modified plaintext $P' = (P'_1, P'_2, P'_3)$. In the next three subparts, write expressions for P'_1, P'_2, and P'_3.

Q1.5 (4 points) P'_1 is equal to these values, XORed together. Select as many options as you need.

For example, if you think $P'_1 = P_1 \oplus C_2$, then bubble in P_1 and C_2.

■ P_1 ■ P_2 □ P_3 ■ IV ■ C_1 □ C_2 □ C_3

Solution: We denote the "original" ciphertext blocks by C_i and the modified ciphertext blocks by C'_i. For example, $C'_1 = C_2$ in our given scheme. This is likewise the case for plaintext blocks.

We have $C_1 = E_K(P_1 \oplus IV)$ and $C_2 = E_K(P_2 \oplus C_1 \oplus P_1)$ from the encryption/decryption formulas.

After swapping, when we decrypt P_1, we plug in C_2's value for C'_1:

$$\begin{aligned} P'_1 &= D_K(C'_1) \oplus IV \\ P'_1 &= D_K(C_2) \oplus IV \\ P'_1 &= D_K(E_K(P_2 \oplus C_1 \oplus P_1)) \oplus IV \\ P'_1 &= P_2 \oplus C_1 \oplus P_1 \oplus IV \end{aligned}$$

Q1.6 (4 points) P'_2 is equal to these values, XORed together. Select as many options as you need.

□ P_1 ■ P_2 □ P_3 □ IV ■ C_1 ■ C_2 □ C_3

Solution: We have $C_1 = E_K(P_1 \oplus IV)$ and $C_2 = E_K(P_2 \oplus C_1 \oplus P_1)$.

We know from the previous subpart that $P'_1 = P_2 \oplus C_1 \oplus P_1\ IV$. Key to this problem is that the decryption formulas will use the "new" values of P', C' for all values since that's what Bob receives/decrypts.

After swapping, when we decrypt P_2, we plug in C_1's value:

$$\begin{aligned} P'_2 &= D_K(C'_2) \oplus P'_1 \oplus C'_1 \\ P'_2 &= D_K(C_1) \oplus P'_1 \oplus C'_1 \\ P'_2 &= D_K(E_K(P_1 \oplus IV)) \oplus P'_1 \oplus C'_1 \\ P'_2 &= (P_1 \oplus IV) \oplus P'_1 \oplus C'_1 \\ P'_2 &= (P_1 \oplus IV) \oplus (P_2 \oplus C_1 \oplus P_1 \oplus IV) \oplus C_2 \\ P'_2 &= P_2 \oplus C_1 \oplus C_2 \end{aligned}$$

Q1.7 (4 points) P_3' is equal to these values, XORed together. Select as many options as you need.

☐ P_1 ☐ P_2 ■ P_3 ☐ IV ☐ C_1 ☐ C_2 ☐ C_3

Solution: We know $P_2' = P_2 \oplus C_1 \oplus C_2$ from the previous subpart and $C_3 = E_K(P_3 \oplus P_2 \oplus C_2)$

Plug in decryption formula for P_3:

$$
\begin{aligned}
P_3' &= D_K(C_3') \oplus P_2' \oplus C_2' \\
P_3' &= D_K(C_3) \oplus P_2' \oplus C_2' \\
P_3' &= D_K(E_K(P_3 \oplus P_2 \oplus C_2)) \oplus P_2' \oplus C_2' \\
P_3' &= (P_3 \oplus P_2 \oplus C_2) \oplus (P_2 \oplus C_1 \oplus C_2) \oplus C_1 \\
P_3' &= P_3
\end{aligned}
$$

This turns out to be an unintended side effect of PCBC (and not a very good one).

Q2 *Cryptography: All or Nothing Security* **(20 points)**

EvanBot decides to modify AES-CTR in order to provide **all-or-nothing security**. All-or-nothing security means that modifying *any* part of the ciphertext will make the *entire* plaintext decrypt to some sort of "garbage" output.

EvanBot designs the following scheme to encrypt $M = (M_1, M_2, ..., M_n)$:

1. EvanBot generates a new random key K_2 on top of the original key K_1. Note that K_2 is **not** known to the decryptor, even though K_1 is.
2. EvanBot transforms M into "pseudo-message" M' by setting $M'_i = M_i \oplus E_{K_2}(i)$.
3. EvanBot adds the block $M'_{n+1} = H(M'_{n+1} \oplus 1) \oplus H(M'_2 \oplus 2) \oplus ... \oplus H(M'_n \oplus n) \oplus K_2$
4. EvanBot derives the ciphertext $C = \mathsf{Enc}(K_1, M')$ using AES-CTR with key K_1 and IV IV.

First, we will walk through the decryption process for this all-or-nothing scheme. Fill in the blanks for the following by answering the multiple-choice subparts below:

1. CodaBot receives C.
2. CodaBot decrypts C with key K_1 to recover _____ (Q2.1).
3. CodaBot sets $K_2 = M'_{n+1} \oplus$ _____ (Q2.2).
4. CodaBot finds i-th original message block as $M_i =$ _____ (Q2.3).

Q2.1 (2 points) Select the correct option for the blank on Step 2:

- ○ K_2
- ○ $M'_i \oplus E_{K_2}(i)$
- ○ $H(M'_1 \oplus 1) \oplus ... \oplus H(M'_n \oplus n)$
- ● M'

Solution: We first need to decrypt the ciphertext C, which decrypts to M' (the pseudo-message) as stated in Step 4 of the encryption process.

Q2.2 (2 points) Select the correct option for the blank on Step 3:

- ○ K_2
- ○ $M'_i \oplus E_{K_2}(i)$
- ● $H(M'_1 \oplus 1) \oplus ... \oplus H(M'_n \oplus n)$
- ○ M'

Solution: We now need to recover K_2 in order to decrypt the pseudo-message into the real message. By re-arranging the formula from Step 3 of the encryption process, we find that $K_2 = M'_{n+1} \oplus H(M'_1 \oplus 1) \oplus ...H(M'_n \oplus n)$.

Q2.3 (2 points) Select the correct option for the blank on Step 4:

- ○ K_2
- ● $M'_i \oplus E_{K_2}(i)$
- ○ $H(M'_1 \oplus 1) \oplus ... \oplus H(M'_n \oplus n)$
- ○ M'

Solution: We can now recover the real message by XOR-ing out $E_{K_2}(i)$ with the i-th block per Step 2 of the encryption process.

Q2.4 (5 points) Explain how modifying an arbitrary ciphertext block prevents recovery of ***any block*** of the original message.

HINT: Show that we cannot recover K_2 if any ciphertext block is modified.

Solution: Say we modify some C_i to C_i'. We then decrypt M_i' (the i-th pseudo-message block) to some gargabe M_i^{*i}.

Recall that we recover K_2 by XOR-ing the hashes of all M_i' with the last ciphertext block. Therefore, since one of the inputs to these hashes is wrong, the entire XOR will be irrecoverably incorrect, since a small change in a hash input will lead to a wildly different output (avalanche effect). This is important to note, because otherwise an attacker could predictably modify the ciphertexts to cancel out their differences and recover the same K_2 (see next subpart).

Q2.5 (5 points) EvanBot wonders if it's really necessary to have the hash function used in Step 3, and decides to replace Step 3 with this new step:

3. EvanBot adds the block $M_1' \oplus 1) \oplus (M_2' \oplus 2) \oplus ... \oplus (M_n' \oplus n) \oplus K_2$ to the end of M'.

Show that it is possible to tamper with the order of the message blocks, i.e. by swapping two blocks. Note that "tamper" means the message will be decrypted to something different, but not all blocks will turn to garbage (i.e. not "all or nothing").

Solution: Say we swap M_1' and M_2'. When decrypting, the client will then *successfully* compute K_2 with the expression above.

Since we are using AES-CTR, we decrypt $M_1 = E_K(IV+1) \oplus C_2$ and $M_2 = E_K(IV+2) \oplus C_1$. Note that the C_1, C_2 in the decryption equations are swapped since we swapped the ciphertext. We then see that (since XOR is commutative):

$$\begin{aligned}&((E_K(IV+1) \oplus C_2) \oplus 1) \oplus ((E_K(IV+2) \oplus C_1) \oplus C_2)... \\ &= ((E_K(IV+1) \oplus C_1) \oplus 1) \oplus ((E_K(IV+2) \oplus C_2) \oplus 2)... \\ &= (M_1' \oplus 1) \oplus (M_2' \oplus 2)...\end{aligned}$$

This does not hold with the hash version, since the inputs to the hash changing even a little bit change the output dramatically (i.e. the XOR does not commute through the hash function).

Q2.6 (4 points) Does the original all-or-nothing scheme (from the beginning of the question) provide integrity?

○ Yes ● No

Explain why or why not.

Solution: This scheme does not provide integrity, since we cannot **detect tampering**. The all-or-nothing property just causes them to decrypt garbage, but this is not sufficient to provide integrity. For example, tampering with a normal AES ciphertext (without MAC) also causes them to decrypt a (at least partially) garbage message, but does not provide integrity.