CS 161 Spring 2025 — Introduction to Computer Security — Exam Prep 5

Q1 *The Red Hood* (15 points)

Jason Todd decides to launch a communications channel in order to securely communicate with the Red Hood Gang over an insecure channel. Jason wants to test different schemes in his attempt to attain confidentiality and integrity.

Notation:

- M is the message Jason sends to the recipient.
- K_1, K_2, and K_3 are secret keys known to only Jason and the recipient.
- ECB, CBC, and CTR represent block cipher encryption modes for a secure block cipher.
- Assume that CBC and CTR mode are called with randomly generated IVs.
- H is SHA2, a collision-resistant, one-way hash function.
- HMAC is the HMAC construction from lecture.

Decide whether each scheme below provides confidentiality, integrity, both, or neither. For all question parts, the ciphertext is the value of C; t **is a temporary value that is not sent as part of the ciphertext.**

Q1.1 (3 points)

$$t = \mathsf{CBC}(K_1, M) \qquad C_1 = \mathsf{ECB}(K_2, t) \qquad C_2 = \mathsf{HMAC}(K_3, t) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ● Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

Solution: This is a typical encrypt-then-MAC scheme with a twist: Instead of including the ciphertext t directly, the ciphertext (but not the MAC) is additionally encrypted with ECB mode. Even though both the HMAC and ECB leak information about t, t doesn't leak information about the plaintext, so the scheme is confidential. The HMAC over t ensures that the input passed to CBC decryption can't be tampered with, so the scheme maintains integrity.

Q1.2 (3 points)

$$t = \mathsf{ECB}(K_1, M) \qquad C_1 = \mathsf{CBC}(K_2, t) \qquad C_2 = \mathsf{HMAC}(K_3, t) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ● Integrity only
- ○ Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

Solution: Notice that t leaks information about the message because it uses insecure ECB mode. C_2 then leaks information about t, which leaks information about the plaintext, so confidentiality is lost (in this case, C_2 is deterministic). However, because the HMAC is computed over t, which is decryptable to the message, integrity is maintained.

Q1.3 (3 points)

$$C_1 = \mathsf{ECB}(K_1, M) \qquad C_2 = H(K_2 \| C_1) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ● Neither confidentiality nor integrity

Solution: C_1 leaks information about M it uses insecure ECB mode, so confidentiality is lost. C_2 does not maintain integrity as it vulnerable to length extension attacks—an attacker could forge $C_2' = H(K_2 \| C_1 \| x)$ and $C_1' = C_1 \| x$, which would be accepted by anyone verifying the hash.

Q1.4 (3 points) For this subpart only, assume that i a monotonically, increasing counter incremented per message.

$$C_1 = \mathsf{CTR}(K_1, M) \qquad C_2 = \mathsf{HMAC}(i, H(C_1)) \qquad C = (C_1, C_2)$$

Clarification issued during exam: Assume that the counter, i, starts at 0.

- ● Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

Solution: Because i is a known value, the key to the HMAC can be predicted, and the scheme does not maintain integrity. However, since the ciphertext is encrypted with secure CTR mode, and the insecure HMAC is computed only over the ciphertext, the scheme maintains confidentiality.

Q1.5 (3 points) For this subpart only, assume that the block size of block cipher is n, the lengths of K_1 and K_2 are n, the length of M must be $2n$, and the length of the hash produced by H is $2n$.

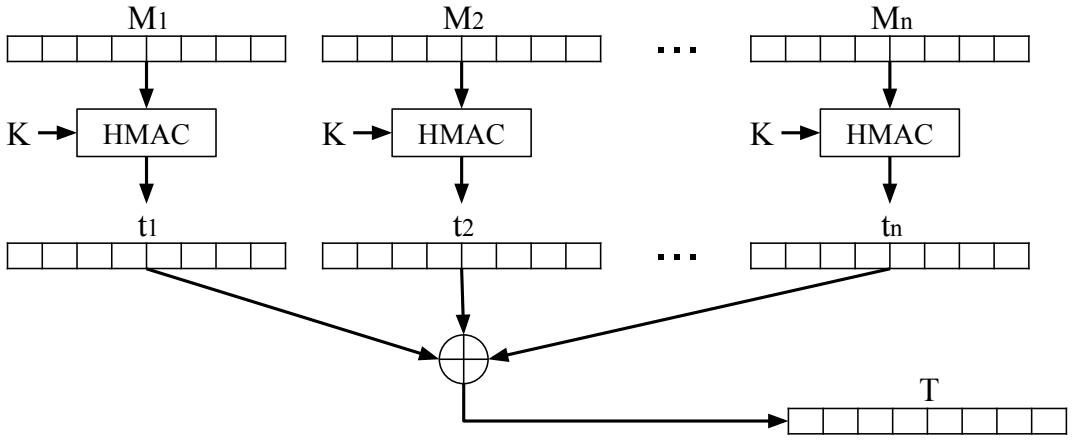
$$C_1 = \mathsf{CBC}(K_1, K_2) \qquad C_2 = M \oplus C_1 \oplus H(C_1) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ● Neither confidentiality nor integrity

Solution: Notice that the attacker already knows the value of C_1 since it is sent with the ciphertext. Because of this, the adversary can just compute $H(C_1)$ then $C_2 \oplus C_1 \oplus H(C_1)$ in order to recover M, so the scheme is not confidential. Additionally, there is no MAC, so the scheme does not have integrity.

Q2 *Integrity and Authenticity: Mix-and-MAC* **(22 points)**

Alice designs a scheme that generates a single MAC on a list of n messages $M_1, M_2, \ldots, M_n$.



1. Compute HMACs on each individual message. $t_i = \mathsf{HMAC}(K, M_i)$, for $1 \leq i \leq n$.
2. XOR all the HMAC outputs (t_i) together to get the final MAC output. $T = t_1 \oplus t_2 \oplus \ldots \oplus t_n$.

Q2.1 (2 points) Does this scheme require the message length to be less than or equal to the length of the HMAC output?

- ○ Yes, because HMAC processes messages one block at a time.
- ○ Yes, because XOR cannot be done between two different-length bitstrings.
- ○ No, because HMAC pads shorter messages to the block length.
- ● No, because HMAC takes in arbitrary-length inputs and outputs fixed-length outputs.

Solution: By definition, HMAC can take in arbitrary-length inputs.

Option (C) is false. HMAC does not pad shorter messages.

Q2.2 (2 points) Alice computes the MAC for the message list $[M_1, \ldots, M_n]$. She sends the message list and the MAC to Bob.

Bob adds a new message M_{n+1} to the list, and wants to compute the MAC of the new message list $[M_1, \ldots, M_n, M_{n+1}]$.

What is the minimum number of HMACs that Bob needs to compute in order to compute the MAC of the new message list?

○ 0 ● 1 ○ 2 ○ $n/2$ ○ n ○ $n+1$

Solution: Bob computes t_{n+1}, which only takes one HMAC to compute. Then Bob can compute $T \oplus t_{n+1}$ to get the MAC of the new list.

Q2.3 (4 points) Alice computes the MAC for two message lists:

- The list $A = [A_1, A_2, \ldots, A_n]$ has MAC T_A.
- The list $B = [B_1, B_2, \ldots, B_n]$ has MAC T_B.

Mallory observes both message lists and both MACs. Mallory does not know K.

Mallory wants to compute a valid MAC on some message list that is not A or B.

Give a valid (message list, MAC) pair that Mallory could compute.

The message list is:

Solution: $[A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n]$

This message list contains all the messages in A and B.

The MAC on the above message list is:

Solution: $T_A \oplus T_B$

This tag can be computed without knowing K. Solutions that try to use K in their expression for the tag would be incorrect, because Mallory does not know K.

The MAC of the combined list of messages is:

$$\mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n) \oplus \mathsf{HMAC}(K, B_1) \ldots \oplus \mathsf{HMAC}(K, B_n)$$

But we know that

$$T_A = \mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n)$$
$$T_B = \mathsf{HMAC}(K, B_1) \oplus \ldots \oplus \mathsf{HMAC}(K, B_n)$$

So we can simplify the MAC expression to $T_A \oplus T_B$.

Note: Answers where the MAC is an expression in terms of K were graded as incorrect, because Mallory does not know K (and would not be able to compute such a MAC).

Q2.4 (4 points) Mallory does not know K. Mallory wants to compute a valid MAC on [pancake], which is a list containing only one message (namely "pancake").

Mallory is allowed to ask for the MAC of two message lists that are not the list [pancake], and Alice will provide the correct MACs for each of the message lists.

The first message list that Mallory queries for is:

Solution: Many alternative solutions exist for this subpart, but most should follow a similar idea to the following.

[pancake, waffle]

"waffle" can be replaced with any arbitrary message in this solution.

Alice reports that the MAC of the message list in the box above is T_1.

The second message list that Mallory queries for is:

Solution: [waffle]

Alice reports that the MAC of the message list in the box above is T_2.

Now, Mallory can compute that the MAC of the message list [pancake] is:

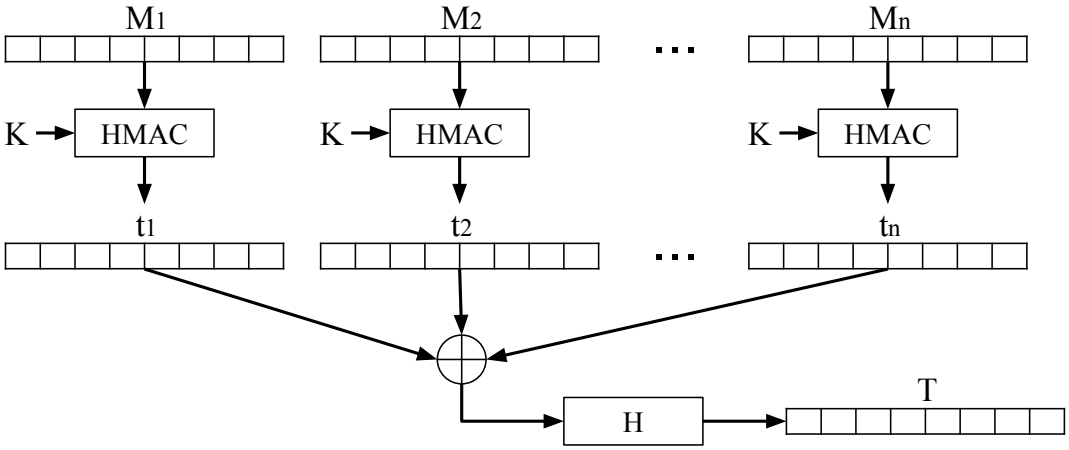
Solution: $T_1 \oplus T_2$

$T_1 = \mathsf{HMAC}(K, \text{pancake}) \oplus \mathsf{HMAC}(K, \text{waffle})$

$T_2 = \mathsf{HMAC}(K, \text{waffle})$

If we XOR these two tags together, then the $\mathsf{HMAC}(K, \text{waffle})$ cancels, leaving us with just $\mathsf{HMAC}(K, \text{pancake})$, as desired.

In the next two subparts, Alice modifies her scheme by adding an extra hashing step at the end:



1. Compute HMACs on each individual message. $t_i = \mathsf{HMAC}(K, M_i)$, for $1 \leq i \leq n$.
2. XOR all the HMAC outputs (t_i) together, **and hash the result**, to get the final MAC output. $T = \mathsf{H}(t_1 \oplus t_2 \oplus \ldots \oplus t_n)$.

Q2.5 (2 points) Using this new scheme, Alice computes the MAC for the message list $[M_1, \ldots, M_n]$. She sends the message list and the MAC to Bob.

Bob adds a new message M_{n+1} to the list, and wants to compute the MAC of the new message list $[M_1, \ldots, M_n, M_{n+1}]$.

What is the minimum number of HMACs that Bob needs to compute in order to compute the MAC of the new message list?

○ 0 ○ 1 ○ 2 ○ $n/2$ ○ n ● $n+1$

Solution: Given T, the output of a hash, there is no way for Bob to learn what the input to the hash was (the XOR of the $t_1, \ldots, t_n$). Therefore, Bob would have to recompute $t_1, \ldots, t_n$ from scratch, and also compute the new t_{n+1}, for a total of $n + 1$ HMAC computations.

Q2.6 (2 points) Does the attack in the third subpart still work with this new scheme?

- ○ Yes, with no modifications.
- ○ Yes, if we apply H to the MAC produced by the attack.
- ○ No, because Mallory cannot compute the hash without knowing K.
- ● No, because the hash function is one-way.

Solution: Mallory knows the two message lists and MACs:

$$T_A = H(\mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n))$$

$$T_B = H(\mathsf{HMAC}(K, B_1) \oplus \ldots \oplus \mathsf{HMAC}(K, B_n))$$

If we try to XOR these two tags together, we would get:

$$H(\mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n)) \oplus H(\mathsf{HMAC}(K, B_1) \oplus \ldots \oplus \mathsf{HMAC}(K, B_n))$$

Which is not the same as the tag on the combined message list:

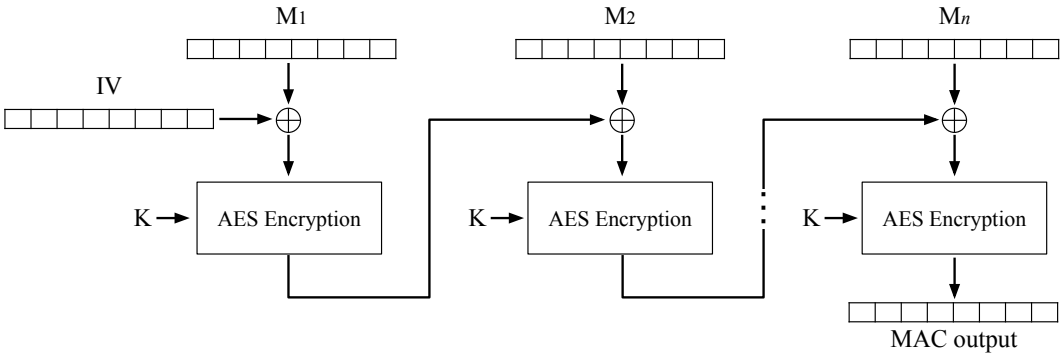
$$H(\mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n) \oplus \mathsf{HMAC}(K, B_1) \oplus \ldots \oplus \mathsf{HMAC}(K, B_n))$$

Option (B) is incorrect because even if we hashed the XOR of the two tags, we would still not get the correct tag on the combined message list.

In order to get from the XORed tags to the tag of the combined message, we would somehow have to invert the hashes to recover the XOR of the HMACs, $\mathsf{HMAC}(K, A_1) \oplus \ldots \oplus \mathsf{HMAC}(K, A_n)$, but this is not possible because the hash function is one-way.

Option (C) is false because Mallory can compute the hash without knowing K.

For the rest of the question, consider this scheme for computing a single MAC on a list of n messages $M_1, M_2, \ldots, M_n$. For the rest of the question, you may assume each message is exactly one block long.



$$T = E_K(M_n \oplus E_K(\ldots M_2 \oplus E_K(M_1 \oplus \text{IV})))$$

The final MAC output is (T, IV).

Q2.7 (2 points) Select all true statements about the scheme above.

☐ Given the list $[M_1, \ldots, M_n]$ and its MAC, it is possible to compute the MAC of list $[M_1, \ldots, M_n, M_{n+1}]$ without knowing K.

☐ The MAC of list $[M_1, M_2, M_3]$ is equal to the MAC of list $[M_3, M_2, M_1]$.

■ None of the above.

Solution:

Option (A) is false. If we want to compute the MAC of the list with an extra message, we need to compute an AES encryption E_K, but we can't do that unless we know K.

Option (B) is false. Each AES encryption scrambles the input unpredictably, so the order that we chain the inputs into the AES encryption blocks affects the final output value.

Q2.8 (2 points) Suppose that you know the MAC of list $[M_1, M_2, M_3]$ and the MAC of list $[M_4, M_5, M_6]$. You want to compute the MAC of the merged list $[M_1, M_2, \ldots, M_6]$. Select all true statements below.

☐ If you know the individual messages $M_1, M_2, \ldots, M_6$, you can compute the merged MAC without knowing K.

☐ If you know K, you can compute the merged MAC without knowing the individual messages.

■ None of the above.

Solution: In order to chain the messages together, we would need to perform additional computation to account for the fact that the two individual MACs have two different IVs, which requires knowing both K and the individual messages.

Q2.9 (2 points) You receive the MAC (T, IV) of the message list $[M_1, M_2, M_3]$. You want to compute a MAC (T', IV) on the new message list $[M_1, M_2, M_3, M_4]$. Select the correct expression for T'.

○ $T' = E_K(T) \oplus M_4$

● $T' = E_K(T \oplus M_4)$

○ $T' = D_K(T) \oplus M_4$

○ $T' = D_K(T \oplus M_4)$

Solution: Looking at the diagram, we need to take the latest AES output, namely T, and chain it forward to the next message. We take the next message M_4, XOR it with T, and compute the AES encryption on the result.

This could also be derived from the equation:

$$T = E_K(M_3 \oplus E_K(M_2 \oplus E_K(M_1 \oplus IV)))$$

The expression we want is

$$E_K(M_4 \oplus E_K(M_3 \oplus E_K(M_2 \oplus E_K(M_1 \oplus IV))))$$

We can substitute in T to get

$$E_K(M_4 \oplus T)$$

as desired.

Q3 *Bonsai* (10 points)

EvanBot wants to store a file in an *untrusted* database that the adversary can read and modify.

Before storing the file, EvanBot computes a hash over the contents of the file and stores the hash separately. When retrieving the file, EvanBot re-computes a hash over the file contents, and, if the computed hash doesn't match the stored hash, then EvanBot concludes that the file has been tampered with.

Clarification during exam: Assume that EvanBot does not know if hashes or files have been modified in the untrusted datastore.

Q3.1 (4 points) What assumptions are needed for this scheme to guarantee integrity on the file? Select all that apply.

■ (A) An attacker cannot tamper with EvanBot's stored hash

☐ (B) EvanBot has a secret key that nobody else knows

☐ (C) The file is at most 128 bits long

■ (D) EvanBot uses a secure cryptographic hash

☐ (E) None of the above

☐ (F) ——

Solution: In order to guarantee integrity on this file, we need two assumptions to hold.

First, the attacker shouldn't be able to tamper with the stored hash. If they could, then the attacker could simply replace the file with an arbitrary file of the attacker's choice, and replace the original stored hash with a hash over this new file. EvanBot's check on the file would succeed.

If EvanBot had a secret key, then EvanBot could change the scheme to use a MAC using the secret key instead of a hash. However, since this scheme uses a hash, a secret key doesn't help us here.

The file being 128 bits long has no relevance to this question.

Finally, the hash must be a secure cryptographic hash. A quick counterexample: if EvanBot used a hash function that mapped every input to the hash value "1", then the attacker could choose an input of their choice, and the check on the hash would always succeed.

For the rest of this question, we refer to two databases: a *trusted database* that an adversary cannot read or modify, and an *untrusted database* that an adversary can read and modify.

Assume that H is a secure cryptographic hash function and $\|$ denotes concatenation.

EvanBot creates and stores four files, F_1, F_2, F_3, and F_4, in the untrusted database. EvanBot also computes and stores a hash on each file's contents in the untrusted database:

$$h_1 = \mathsf{H}(F_1) \qquad h_2 = \mathsf{H}(F_2) \qquad h_3 = \mathsf{H}(F_3) \qquad h_4 = \mathsf{H}(F_4)$$

Then, EvanBot stores $h_{\text{root}} = \mathsf{H}(h_1 \| h_2 \| h_3 \| h_4)$ in the *trusted* database.

Q3.2 (3 points) If an attacker modifies F_2 stored on the server, will EvanBot be able to detect the tampering?

● (G) Yes, because EvanBot can compute h_{root} and see it doesn't match the stored h_{root}

○ (H) Yes, because EvanBot can compute h_2 and see it doesn't match the stored h_2

○ (I) No, because the hash doesn't use a secret key

○ (J) No, because the attacker can re-compute h_2 to be the hash of the modified file

○ (K) —

○ (L) —

Solution:

In this scheme, we have a trusted database that an adversary cannot read or modify. Because we have this trusted database, it's possible to ensure integrity through the use of hashes, despite them not being signed (like MAC's).

Let's walk through what happens if an attacker modifies F_2. If the attacker modifies this file and nothing else, then it's easy for Bot to detect tampering: Bot just has to recompute a hash over F_2 and realize that it doesn't match h_2.

However, an attacker can also modify h_2 to be the hash of the malicious file, since it's in the untrusted database. Because of this, in order to detect tampering, Bot has to use the only thing that the attacker doesn't have access to: h_{root}, which is stored in the trusted database.

Based on this information: the simplest way to verify the integrity of F_2 is to:

1. Recompute a hash over F_1, F_2, F_3, and F_4.
2. Recompute h_{root} using these hashes.
3. Compare this h_{root} to the stored version of h_{root}.

If the attacker modifies F_2, then Bot will **always** be able to detect the tampering, since the check on the root hashes will fail.

Q3.3 (3 points) What is the minimum number of hashes EvanBot needs to compute to verify the integrity of all four files?

◯ (A) 1 ◯ (C) 3 ● (E) 5

◯ (B) 2 ◯ (D) 4 ◯ (F) More than 5

Solution:

Because the attacker has the ability to modify all files and hashes in the insecure database, Bot needs to make sure that the attacker hasn't modified any single file/hash pair. To do this, Bot need to follow the procedure discussed in Q3.2's solution - recompute a hash over each file (4 hashes in total), and recompute the root hash (1 hash in total).