CS 161 Fall 2023 — Introduction to Computer Security — Midterm

Name: _____

Student ID: _____

This exam is 110 minutes long.

Question:	1	2	3	4	5	6	7	Total
Points:	1	13	14	17	16	22	17	100

For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)
- ⊘ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

- ■ You can select
- ■ multiple squares (completely filled)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation.

Pre-exam activity:

(Just for fun, not graded.)

Let's play Rock Paper Scissors with EvanBot! Circle your move below, and we will reveal your result after the exam!

Rock Paper Scissors

To prove EvanBot isn't cheating, here is the SHA256 hash of EvanBot's move:

```
406dba28c82f3d8ff18b2df401d5b02c
6b1debc42ca5866585497fdb23e51741
```

Q1 *Honor Code* (1 point)

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam.

Read the honor code above and sign your name: _____

Q2 *True/False* **(13 points)**

Each true/false is worth 1 point.

Q2.1 EvanBot brings both an umbrella and a raincoat, so that EvanBot will stay dry even if the raincoat tears or the umbrella breaks.

TRUE or FALSE: This is an example of defense in depth.

◯ (A) TRUE ◯ (B) FALSE

Q2.2 A store installs sensors at the main exit to detect shoplifting, but there is another exit in the back of the store without sensors installed.

TRUE or FALSE: This is a failure to ensure complete mediation.

◯ (A) TRUE ◯ (B) FALSE

For the next 3 subparts: Suppose we have a little-endian C program with a local variable `char buf[64]`. Consider the following possible GDB output after running the command `x/8wx buf`:

```
0xffff1430: 0x00003500 0x6ca59820 0x0000abcd 0x74392bc2
0xffff1434: 0x00000000 0x00000000 0x00000000 0x00000000
```

Q2.3 TRUE or FALSE: The addresses in the left-most column of this GDB output are valid GDB output.

◯ (A) TRUE ◯ (B) FALSE

Q2.4 TRUE or FALSE: `buf[5]` is `0x98`.

◯ (A) TRUE ◯ (B) FALSE

Q2.5 TRUE or FALSE: The command `x/16wx buf` would display all 64 bytes in `buf`.

◯ (A) TRUE ◯ (B) FALSE

Q2.6 TRUE or FALSE: A C program that never lets the user change the values of the RIP and SFP is safe against all memory safety attacks.

◯ (A) TRUE ◯ (B) FALSE

Q2.7 TRUE or FALSE: A 128-bit stack canary would be secure against any practical brute-force attacks against the canary.

◯ (A) TRUE ◯ (B) FALSE

Q2.8 TRUE or FALSE: When pointer authentication is enabled, an attacker who inspects the program memory cannot learn the values of any addresses.

○ (A) TRUE ○ (B) FALSE

Q2.9 You are using GDB to debug a program with ASLR enabled.

TRUE or FALSE: Running a valid `x` command (e.g. `x/4wx var` for some local variable `var`) will always help you learn something about the address randomization on the **current** run of the program.

○ (A) TRUE ○ (B) FALSE

Q2.10 You are using GDB to debug a program with ASLR enabled.

TRUE or FALSE: Running a valid `x` command will always help you learn something about the address randomization on a **different** run of the program.

○ (A) TRUE ○ (B) FALSE

Q2.11 TRUE or FALSE: Publicly revealing the value of nothing-up-my-sleeve numbers does not compromise the security of the cryptographic system.

○ (A) TRUE ○ (B) FALSE

Q2.12 Alice decides to build a PRNG with a secure hash function H. For a seed s, she outputs $H(s\|0)$ and sets the new seed to $H(s\|1)$.

TRUE or FALSE: This PRNG is rollback-resistant.

○ (A) TRUE ○ (B) FALSE

Q2.13 TRUE or FALSE: Creating a new certificate requires generating a secure digital signature.

○ (A) TRUE ○ (B) FALSE

Q3 *Memory Safety: Homecoming* **(14 points)**

Consider the following vulnerable C code:

```
1  void vulnerable() {
2      char vulnerable_buf[32];
3      fgets(vulnerable_buf, 32, stdin);
4      helper(vulnerable_buf);
5  }
6
7  void helper(char* arg) {
8      char helper_buf[32];
9      fgets(helper_buf, 37, stdin);
10 }
```

These assembly instructions occur in memory:

```
1 0x08076000: ret
2 0x08076004: pop %eax
3 ...
4 0x08076030: call helper
5 0x08076034: add $4, %esp
6 0x08076038: mov %ebp, %esp
7 0x0807603c: pop %ebp
8 0x08076040: ret
```

Stack at Line 8

RIP of `vulnerable`
(1)
(2)
(3)
(4)
(5)
(6)

Assumptions:

- You may use `SHELLCODE` as a 31-byte shellcode.
- ASLR is enabled, **not** including the code segment (i.e. the addresses in the code segment don't change).
- Using GDB, you find that the RIP of `helper` has value `0x08076034`.
- Unless otherwise specified, all other memory safety defenses are disabled.

Q3.1 (1 point) What values go in blanks (1) through (3) in the stack diagram above?

- ○ (A) (1) `arg` (2) SFP of `vulnerable` (3) `vulnerable_buf`
- ○ (B) (1) SFP of `vulnerable` (2) `arg` (3) `vulnerable_buf`
- ○ (C) (1) SFP of `vulnerable` (2) `vulnerable_buf` (3) `arg`
- ○ (D) (1) `vulnerable_buf` (2) SFP of `vulnerable` (3) `arg`

Q3.2 (1 point) What values go in blanks (4) through (6) in the stack diagram above?

- ○ (A) (4) `helper_buf` (5) RIP of `helper` (6) SFP of `helper`
- ○ (B) (4) RIP of `helper` (5) `helper_buf` (6) SFP of `helper`
- ○ (C) (4) RIP of `helper` (5) SFP of `helper` (6) `helper_buf`
- ○ (D) (4) SFP of `helper` (5) `helper_buf` (6) RIP of `helper`

Q3.3 (2 points) Recall the off-by-one exploit in Project 1, where we overwrite an SFP with a specific value that causes shellcode to execute.

Why does the off-by-one exploit not work here with 100% probability?

- ○ (A) ASLR is enabled, so we don't know what value to overwrite an SFP with.
- ○ (B) ASLR is enabled, so instructions written on the stack are not executable.
- ○ (C) The off-by-one exploit requires two returns after overwriting an SFP, and this code only returns once after overwriting an SFP.
- ○ (D) It is not possible to overwrite an SFP in this code.

Here is an exploit that causes shellcode to execute with 100% probability:

Input to `fgets` in `vulnerable`: `SHELLCODE`

Input to `fgets` in `helper`: `SHELLCODE + 5*'A'`

Q3.4 (2 points) The function epilogue for `helper` runs:

```
mov %ebp, %esp
pop %ebp
ret
```

What is the next instruction executed after `ret`?

- ○ (A) `pop %eax`
- ○ (B) `pop %ebp`
- ○ (C) The `ret` instruction at `0x08076000`
- ○ (D) The `ret` instruction at `0x08076040`
- ○ (E) The `SHELLCODE` in `vulnerable_buf`
- ○ (F) The `SHELLCODE` in `helper_buf`

Q3.5 (2 points) What is the next instruction executed after your answer to Q3.4?

- ○ (A) `pop %eax`
- ○ (B) `pop %ebp`
- ○ (C) The `ret` instruction at `0x08076000`
- ○ (D) The `ret` instruction at `0x08076040`
- ○ (E) The `SHELLCODE` in `vulnerable_buf`
- ○ (F) The `SHELLCODE` in `helper_buf`

Q3.6 (2 points) For this subpart only, suppose we run this exploit on a **big-endian** system instead of a little-endian system.

Does the exploit listed above still work?

- ○ (A) Yes, because the same bytes are written into memory.
- ○ (B) Yes, because endianness does not affect shellcode (the instructions are always executed from low to high addresses).
- ○ (C) No, because a different byte of the RIP is overwritten.
- ○ (D) No, because endianness affects shellcode (we would have to reverse the shellcode to make the exploit work).

Q3.7 (2 points) For this subpart only, suppose that we swap the `ret` instruction at address `0x08076000` and the `pop %eax` instruction at address `0x08076004`.

Does the exploit listed above still work?

- ○ (A) Yes, with no modifications.
- ○ (B) Yes, but only if we replace `'A'*5` with `'\x04'*5`.
- ○ (C) No, because we cannot change the last byte that `fgets` writes in memory.
- ○ (D) No, because `pop %eax` is not a valid x86 assembly instruction.

Q3.8 (2 points) For this subpart only, suppose that we run this code on a 64-bit system, with pointer authentication codes (PACs) enabled.

Also, suppose that `fgets(helper_buf, 37, stdin)` is changed to `fgets(helper_buf, 41, stdin)`.

Does the exploit listed above still work?

- ○ (A) Yes, because the exploit never changes the value of `arg` in memory.
- ○ (B) Yes, because the exploit modifies addresses without overwriting any PACs.
- ○ (C) No, because PACs prevent all buffer overflows.
- ○ (D) No, because the exploit changes the value of an address in memory.

Q4 *Memory Safety: Forbidden Instruction* **(17 points)**

Consider the following vulnerable C code:

```
1 void vulnerable() {
2     char buf[8];
3     gets(buf);
4 }
```

Assumptions:

- Each x86 instruction is 4 bytes long in machine code.
- The address of `buf` is `0xffff1230`.
- In the entire question, the `SHELLCODE` we want to execute is 16 bytes long.
- The system runs a special variant of non-executable pages: the instruction that assembles to the machine code `0xdeadbeef` cannot be executed on any page marked as non-executable. All other instructions can be executed anywhere in memory.

Q4.1 (3 points) For this subpart only, suppose that the 16-byte shellcode appears in the code section of memory at address `0x10101234`.

Select all inputs to the `gets` call that would cause the shellcode to execute. (Assume every input is followed by a newline character.)

- ☐ (A) `'A' * 8 + '\x34\x12\x10\x10'`
- ☐ (B) `'A' * 12 + '\x34\x12\x10\x10'`
- ☐ (C) `'\x34\x12\x10\x10' * 4`
- ☐ (D) `'A' * 12 + '\x10\x10\x12\x34'`
- ☐ (E) `'A' * 8 + '\x10\x10\x12\x34'`
- ☐ (F) None of the above.

In the next few subparts, suppose that only the *last* instruction of shellcode is the special instruction. In other words, `SHELLCODE[12:16] = 0xdeadbeef`. All other instructions in the shellcode are not the special instruction.

Q4.2 (2 points) Which sequence of bytes must exist in the code section of memory in order for us to execute shellcode?

- ○ (A) `0x00000000`
- ○ (B) `0xdeadbeef`
- ○ (C) `0xffff1230`
- ○ (D) `0xffff123c`
- ○ (E) A `nop` instruction
- ○ (F) A `ret` instruction

Q4.3 (2 points) Assume that the byte sequence you provided in the previous subpart appears in the code section of memory at address `0x10101234`.

In addition to shellcode, the exploit will need to contain a single extra x86 instruction. What should this extra instruction do?

- ○ (A) Jump to the address `0xdeadbeef`.
- ○ (B) Jump to the address `0xffff1230`.
- ○ (C) Jump to the address `0x10101234`.
- ○ (D) The special instruction (`0xdeadbeef`).
- ○ (E) The `ret` instruction.
- ○ (F) The `nop` instruction.

Q4.4 (4 points) Again, assume that the byte sequence you provided in Q4.2 appears at address `0x10101234`.

Provide an input to the `gets` call that would cause the shellcode to execute.

You may use the variable `INST` to represent the 4-byte x86 instruction you answered in the previous subpart.

For the rest of the question, suppose that only the *first* instruction of shellcode is the special instruction. In other words, `SHELLCODE[0:4] = 0xdeadbeef`. All other instructions in the shellcode are not the special instruction.

Also, suppose that the following x86 instructions appear in the code section of memory. (The sequence of bytes you provided in Q4.2 are no longer in the code section of memory.)

```
1 0x10101280: jmp 0x1010129c
2 0x10101284: pop %eax
3 0x10101288: call helper
4 0x1010128c: add $4, %esp
5 0x10101290: mov %esp %ebp
6 0x10101294: 0xdeadbeef (special instruction)
7 0x10101298: ret
8 0x1010129c: 0xdeadbeef (special instruction)
9 0x101012a0: call exit (terminates the program)
```

Q4.5 (6 points) Provide an input to the `gets` call that would cause the shellcode to execute.

Q5 *Symmetric-Key Cryptography: Not Quite by DESign* **(16 points)**

Q5.1 (2 points) Suppose we take AES-CBC encryption and replace all AES encryption blocks with HMAC. The new encryption formula for a message $M = (M_1, M_2, \ldots, M_n)$ is:

$$C_0 = IV$$
$$C_i = \mathsf{HMAC}(K, M_i \oplus C_{i-1})$$

Select all true statements about this new scheme.

- ☐ (A) M must be padded until M is a multiple of the HMAC output size before computing C.
- ☐ (B) Someone who knows K is able to decrypt any ciphertext.
- ☐ (C) Someone who does not know K is able to decrypt any ciphertext.
- ☐ (D) None of the above.

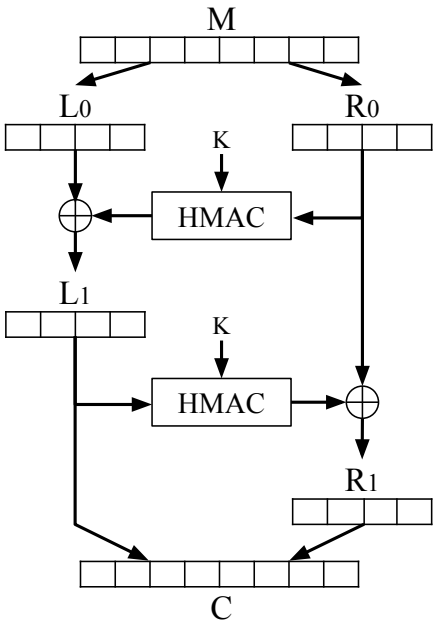
Q5.2 (2 points) Now suppose we take AES-CTR encryption and replace all AES encryption blocks with HMAC. The new encryption formula is:

$$C_i = M_i \oplus \mathsf{HMAC}(K, IV + i)$$

Select all true statements about this new scheme.

- ☐ (A) M must be padded until M is a multiple of the HMAC output size before computing C.
- ☐ (B) Someone who knows K is able to decrypt any ciphertext.
- ☐ (C) Someone who does not know K is able to decrypt any ciphertext.
- ☐ (D) None of the above.

The rest of the question is independent of the first two subparts.



For the rest of the question, consider this scheme for encrypting 128-bit messages:

1. Split the message into two halves.
   $L_0 = M$`[:64]`
   $R_0 = M$`[64:128]`
2. Set $L_1 = L_0 \oplus \mathsf{HMAC}(K, R_0)$.
3. Set $R_1 = R_0 \oplus \mathsf{HMAC}(K, L_1)$.
4. Concatenate L_1 and R_1 to get the ciphertext.
   $C = L_1 \| R_1$

For this question, you can assume that HMAC outputs 64 bits.

A copy of the diagram on the previous page is available on the appendix.

Q5.3 (2 points) What is the decryption formula for L_0?

L_0 is equal to these values, XORed together. Select as many options as you need.

For example, if you think $L_0 = R_1 \oplus L_1$, then bubble in R_1 and L_1.

- ☐ (A) R_0
- ☐ (B) $\mathsf{HMAC}(K, R_0)$
- ☐ (C) R_1
- ☐ (D) $\mathsf{HMAC}(K, R_1)$
- ☐ (E) L_1
- ☐ (F) $\mathsf{HMAC}(K, L_1)$

Q5.4 (2 points) What is the decryption formula for R_0?

R_0 is equal to these values, XORed together. Select as many options as you need.

- ☐ (A) L_0
- ☐ (B) $\mathsf{HMAC}(K, L_0)$
- ☐ (C) R_1
- ☐ (D) $\mathsf{HMAC}(K, R_1)$
- ☐ (E) L_1
- ☐ (F) $\mathsf{HMAC}(K, L_1)$

In the next two subparts, consider this scenario: Alice encrypts a message M and gets ciphertext C. Later, Alice encrypts M again, but with one bit in L_0 flipped.

Q5.5 (2 points) What happens to L_1 in the resulting ciphertext?

- ○ (A) It is unchanged.
- ○ (B) One bit is flipped.
- ○ (C) It is unpredictably different (garbage).
- ○ (D) None of the above.

Q5.6 (2 points) What happens to R_1 in the resulting ciphertext?

- ○ (A) It is unchanged.
- ○ (B) One bit is flipped.
- ○ (C) It is unpredictably different (garbage).
- ○ (D) None of the above.

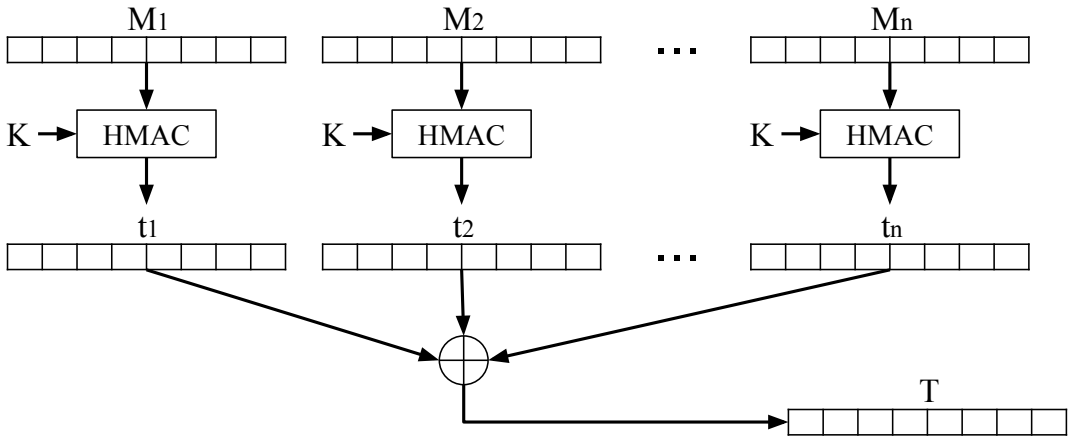
Q5.7 (4 points) Does this scheme provide integrity?

- ○ (A) Yes
- ○ (B) No

Briefly explain your answer (two sentences or fewer is sufficient).

Q6 *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. $T = t_1 \oplus t_2 \oplus \ldots \oplus t_n$.

Q6.1 (2 points) Does this scheme require the message length to be less than or equal to the length of the HMAC output?

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

Q6.2 (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?

○ (A) 0 ○ (B) 1 ○ (C) 2 ○ (D) $n/2$ ○ (E) n ○ (F) $n+1$

A copy of the diagram on the previous page is available on the appendix.

Q6.3 (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:

The MAC on the above message list is:

Q6.4 (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:

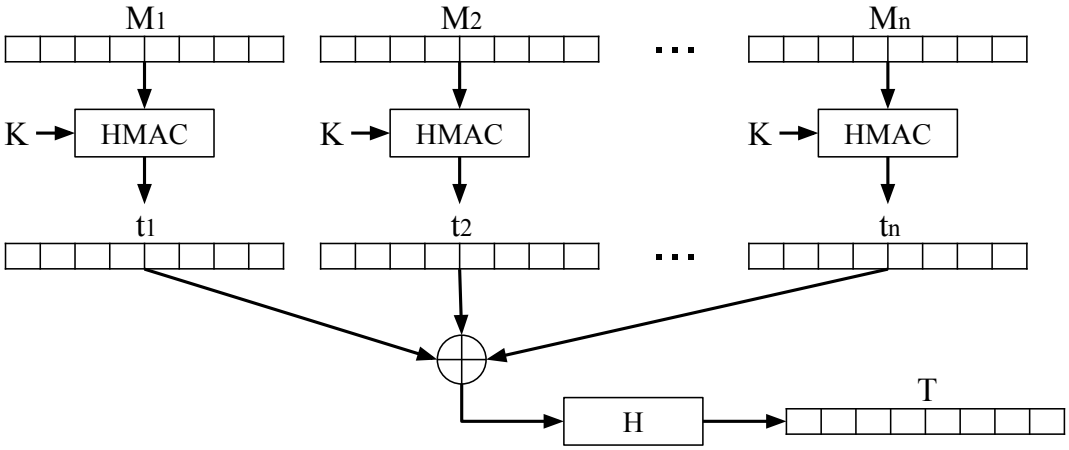
Alice reports that the MAC of the message list in the box above is T_1.

The second message list that Mallory queries for is:

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:

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. $T = \mathsf{H}(t_1 \oplus t_2 \oplus \ldots \oplus t_n)$.

Q6.5 (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?

○ (A) 0 ○ (B) 1 ○ (C) 2 ○ (D) $n/2$ ○ (E) n ○ (F) $n+1$

Q6.6 (2 points) Does the attack in Q6.3 still work with this new scheme?

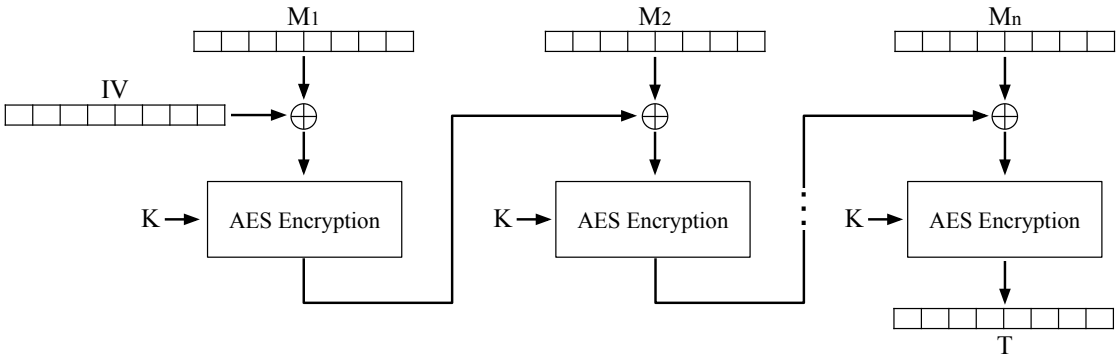
○ (A) Yes, with no modifications.

○ (B) Yes, if we apply H to the MAC produced by the attack.

○ (C) No, because Mallory cannot compute the hash without knowing K.

○ (D) No, because the hash function is one-way.

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).

Q6.7 (2 points) Select all true statements about the scheme above.

- ☐ (A) 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.
- ☐ (B) The MAC of list $[M_1, M_2, M_3]$ is equal to the MAC of list $[M_3, M_2, M_1]$.
- ☐ (C) None of the above.

Q6.8 (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.

- ☐ (A) If you know the individual messages $M_1, M_2, \ldots, M_6$, you can compute the merged MAC without knowing K.
- ☐ (B) If you know K, you can compute the merged MAC without knowing the individual messages.
- ☐ (C) None of the above.

Q6.9 (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'.

- ○ (A) $T' = E_K(T) \oplus M_4$
- ○ (B) $T' = D_K(T) \oplus M_4$
- ○ (C) $T' = E_K(T \oplus M_4)$
- ○ (D) $T' = D_K(T \oplus M_4)$

Q7 *Public-Key Cryptography: Does EvanBot Snore?* **(17 points)**

EvanBot decides to make a new authentication scheme that works over insecure connections.

Assume all parties have agreed on generator g for prime modulus p in advance.

To create a new account:

1. The user generates a random number a as their password and sends $A = g^a \bmod p$.
2. The server stores a mapping between the user and their corresponding A.

To login:

1. The user chooses a random $r \bmod p$ and sends $R = g^r \bmod p$ to the server.
2. The server chooses a random $b \bmod p$ and sends $b \bmod p$ to the user.
3. The user sends $z = r + ab \pmod{p-1}$ to the server.
4. The server checks if [ANSWER TO Q7.1].

Q7.1 (2 points) Fill in the missing value in the last step.

- ○ (A) $g^b \equiv A \cdot R^z \bmod p$
- ○ (B) $g^A \equiv R \cdot b \cdot A \bmod p$
- ○ (C) $g^z \equiv R^b \cdot A \bmod p$
- ○ (D) $g^z \equiv R \cdot A^b \bmod p$

Q7.2 (3 points) Select all true statements.

- ☐ (A) $r + ab \pmod{p-1}$ hides the value of a, even though the server knows b.
- ☐ (B) $R = g^r \bmod p$ hides the value of r.
- ☐ (C) r being chosen randomly prevents replay attacks.
- ☐ (D) None of the above.

Q7.3 (2 points) What happens if r is leaked to an eavesdropping attacker that has seen the full login process?

- ○ (A) The attacker can use r to recover the password a.
- ○ (B) The attacker can tell when two different logins use the same password.
- ○ (C) The attacker can use r to recover b.
- ○ (D) The attacker is able to solve the discrete log problem on logins from different users.

For the rest of this question, consider this modified, non-interactive authentication scheme.

1. The user chooses a random $r \bmod p$ and sets $R = g^r \bmod p$.
2. The user sets $b = H(R)$.
3. The user sends R and $z = r + ab \pmod{p-1}$.
4. The server derives $b = H(R)$.
5. The server checks if [ANSWER TO Q7.1].

Q7.4 (3 points) Which option best explains why this scheme convinces the server that the user knows a, despite b not being a server-provided random input?

- ○ (A) Sending $g^r \bmod p$ forces the user to stick with the same r throughout the protocol.
- ○ (B) The hash function is collision resistant, making it hard to find a different value that hashes to the same b.
- ○ (C) An attacker cannot predict the value of b due to the unpredictability of the hash function.
- ○ (D) The server can check that $b = H(R)$ because the hash function is publicly known.

Q7.5 (3 points) Is this scheme secure against replay attacks?

- ○ (A) Yes, because setting $b = H(R)$ effectively replaces the server's random input.
- ○ (B) Yes, because any change to r will cause $b = H(R)$ to have a wildly different output.
- ○ (C) No, because eavesdroppers can record R and z from the login transcript and resend R, z on their own.
- ○ (D) No, because the value of R is the same between authentications made by the same user.

Q7.6 (4 points) Consider a modified scheme where Step 2 and Step 4 set $b = H(A)$ instead of $b = H(R)$. Provide values for R and z that will validate for any given A.

R:

z:

Q7.7 (0 points) **This A+ question is not worth points. It can only affect your course grade if you have a high A and might receive an A+, however, there are other ways to earn an A+ if you are not able to complete this question. We strongly recommend completing the rest of the exam first. Ask your proctor for additional paper if you need more space to write.**

Design a secure digital signature scheme based on the security of the non-interactive authentication scheme used from Q7.4 onwards. **Your scheme should not significantly deviate from the non-interactive authentication protocol.**

Describe an algorithm to sign a message. The input will be a secret key sk (such that $pk = g^{sk} \bmod p$ for publicly-known g, p) and a message M.

HINT: Use the steps from the non-interactive authentication protocol as a starting point.

Describe an algorithm to verify the signature produced in your given signing algorithm. The input will be a public key pk, signature S, and message M.

Explain why your algorithm is a secure digital signature scheme, assuming the non-interactive protocol used from Q7.4 onwards is secure.

*HINT: You don't need to write a formal proof, but you should explain **precisely** why the changes you made to turn the authentication scheme into a signature scheme are **necessary** and **secure**.*

Nothing on this page will affect your grade.

Post-Exam Activity: Pumpkin

It's spooky season! How will you carve this huge pumpkin?

Comment Box

Congratulations for making it to the end of the exam! Feel free to leave any thoughts, comments, feedback, or doodles here: