CS 161 Spring 2026 | Introduction to Computer Security | Midterm

Name: ______________________________

Student ID: ______________________________

This exam is 110 minutes long. There are 7 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	Total
Points:	0	10	16	22	9	24	19	100

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

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

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

- ■ You can select
- ■ multiple squares (completely filled).
- ☑ (Don't do this)

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 may grade the worst interpretation.

Pre-Exam Activity: (0 points): EvanBot is a hungry food critic! What does bot get to try tonight?

Art by: Soklynin Nou

Q1 *Honor Code* (0 points)

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: ______________________________

This page left intentionally (mostly) blank

The exam continues on the next page.

Q2 *Potpourri* **(10 points)**

Each question is worth 1 point.

Q2.1 EvanBot configures a file-sharing application so that it can only access a specific folder instead of the entire filesystem.

TRUE OR FALSE: This is an example of the principle of least privilege.

○ TRUE ○ FALSE

For Q2.2–Q2.3, consider the C code and GDB output stopped at line 7.

C code:

```
1 void callee() {
2   char buf[16];
3   gets(buf);
4 }
5
6 void caller() {
7   callee();
8 }
```

GDB Output (line 7):

```
info frame
...
Saved registers:
ebp at 0xffffdc58,
eip at 0xffffdc5c
```

GDB Output (line 3):

```
x/1x buf
0xffffdc60: 0xdeadbeef
```

Q2.2 TRUE OR FALSE: Running `print buf` will display the bytes in the local array buf.

○ TRUE ○ FALSE

Q2.3 TRUE OR FALSE: `0xffffdc5c` is the address on the stack where the RIP of `caller` is stored.

○ TRUE ○ FALSE

For Q2.4–Q2.5, consider the C code and GDB output stopped at line 3.

Q2.4 TRUE OR FALSE: `0xffffdc60` is the address of the first byte of `buf`.

○ TRUE ○ FALSE

Q2.5 TRUE OR FALSE: `0xde` is the first byte of `buf`.

○ TRUE ○ FALSE

Q2.6 Which of these properties should a pseudo-random function exhibit? Select all that apply.

☐ Deterministic ☐ Efficient to compute ○ None of the above

☐ Keyed ☐ Randomized on every call

Q2.7 TRUE OR FALSE: If a function is one-way, it is guaranteed to be a collision-resistant hash function.

○ TRUE ○ FALSE

Q2.8 TRUE OR FALSE: An adversary with a polynomial-time factoring algorithm can break the hashed ElGamal encryption scheme.

○ TRUE ○ FALSE

Q2.9 TRUE OR FALSE: Merkle puzzles are more secure than Diffie-Hellman key exchange because they provide a larger asymptotic computational gap between the honest parties and the attacker.

○ TRUE ○ FALSE

Q2.10 TRUE OR FALSE: Deterministic encryption schemes can be secure against chosen-plaintext attacks.

○ TRUE ○ FALSE

Q3 *Memory Safety: Football's Coming Home* (16 points)

Consider the following vulnerable C code:

```
1  void somethingBritish() {
2      int value = 161;
3      char buf[24];
4      fread(buf, 40, 1, stdin);
5      memcpy((buf + value), *(void**)buf, 4);
6      if (value > 32) {
7        exit(0);
8      }
9  }
10
11 void main() {
12     somethingBritish();
13 }
```

Stack at Line 5

RIP of `main`
SFP of `main`
canary
RIP of `somethingBritish`
SFP of `somethingBritish`
(1)
(2)
(3)

Assumptions:

- Stack canaries are enabled and consist of non-null bytes.
- All other memory safety mitigations are disabled.
- You run GDB once and find that the RIP of `somethingBritish` is located at `0xffffdc90`.
- Your goal is to place and execute a 20-byte `SHELLCODE`.

(1 point each) What goes in the blanks (1) - (3) in the stack diagram above?

Q3.1 Blank (1): ◯ `value` ◯ `buf` ◯ canary

Q3.2 Blank (2): ◯ `value` ◯ `buf` ◯ canary

Q3.3 Blank (3): ◯ `value` ◯ `buf` ◯ canary

Q3.4 (2 points) Which of the following memory safety vulnerabilities are present in this code? Select all.

☐ Format string vulnerability ☐ Stack overflow

☐ Signed/unsigned vulnerability ◯ None of the above

Q3.5 (6 points) Provide an input to `fread` on Line 4 that would cause the program to execute `SHELLCODE`. If a part of the input can be any non-zero value, use `'A' * n` to represent `n` bytes of garbage.

Hint: `*(void**)X` *treats the value stored at address* `X` *as a pointer to another location in memory.*

Input to `fread` on line 4:

Q3.6 (3 points) Which changes would cause the correct exploit (without modification) to fail? Consider each choice independently. Select all that apply.

☐ Line 3: Change `char buf[24];` to `char buf[32];`.

☐ Line 2: Change `int value = 161;` to `unsigned int value = 161;`.

☐ Line 4: Swap this line with line 5, so `memcpy` happens before `fread`.

☐ Line 4: Change `fread(buf, 40, 1, stdin);` to `fread(buf, 32, 1, stdin);`.

◯ None of the above

Q3.7 (2 points) Would the correct exploit (without modification) fail if ASLR was enabled?

◯ Yes, because the absolute address of the stack changes every time the program runs.

◯ Yes, because ASLR makes the stack non-executable, preventing the shellcode inside `buf` from running even if the RIP is overwritten.

◯ No, because the relative distance between `buf` and the RIP remains constant regardless of the absolute address.

◯ No, because ASLR only randomizes the location of the heap and shared libraries, not the stack where `buf` is located.

Q4 *Memory Safety: Stephen Curry* (22 points)

Consider the following vulnerable C code:

```
1   void Kyrie() {
2       printf(____________________);
                     Q4.5
3   }
4   void LeBron() {
5       char buf[64];
6       fread(buf, 64, 1, stdin);
7       Kyrie();
8   }
9
10  void main() {
11      LeBron();
12  }
```

Stack at Line 2

RIP of (1)
SFP of (1)
RIP of (2)
SFP of (2)
`buf`
RIP of (3)
SFP of (3)

Assumptions:

- All memory safety defenses are disabled.
- You run GDB once and find that the RIP of `LeBron` is located at `0xffffdc34`.
- Your goal is to place and execute a 32-byte `SHELLCODE`.

(1 point each) What goes in the blanks (1) - (3) in the stack diagram above?

Q4.1 Blank (1): ◯ `main` ◯ `LeBron` ◯ `Kyrie`

Q4.2 Blank (2): ◯ `main` ◯ `LeBron` ◯ `Kyrie`

Q4.3 Blank (3): ◯ `main` ◯ `LeBron` ◯ `Kyrie`

Q4.4 (3 points) What is the **value** (not the address) of the SFP of `Kyrie`?

◯ `0xffffdbe8` ◯ `0xffffdc30` ◯ `0xffffdc38`

◯ `0xffffdbec` ◯ `0xffffdc34` ◯ None of the above

Q4.5 (3 points) What should Line 2 be in order to create a vulnerability that allows the attacker to execute `SHELLCODE`?

Hint: Your reference sheet contains definitions for these format string specifiers.

◯ `printf("%d")` ◯ `printf("%c")` ◯ `printf("%hn")`

◯ `printf("%s")` ◯ `printf("%n")` ◯ `printf("%hhn")`

Q4.6 (2 points) What type of exploit can be constructed as a result of your answer to the previous subpart?

◯ Buffer Overflow ◯ Off-by-one

◯ Signed/unsigned vulnerability ◯ None of the above

Q4.7 (6 points) Provide an input to `fread` on Line 6 that would cause the program to execute `SHELLCODE`, If a part of the input can be any non-zero value, use `'A' * n` to represent `n` bytes of garbage.

Input to `fread` on Line 6:

Q4.8 (2 points) Would the correct exploit (without modification) fail if Non-Executable Pages was enabled?

- ◯ Yes, because the CPU will refuse to execute instructions located on the stack, even if the RIP successfully points to them.
- ◯ Yes, because Non-Executable Pages randomizes the stack addresses, making it impossible to determine the location of the shellcode.
- ◯ No, because Non-Executable Pages only prevents the `RIP` from being overwritten, but does not stop the execution of shellcode.
- ◯ No, because the `fread` function bypasses Non-Executable permissions when writing the shellcode into memory.

Q4.9 (3 points) Which **addresses** of the SFP of `LeBron` would prevent the type of exploit from Q4.7? Assume that the stack is shifted up/down accordingly and remains in-order. Select all that apply.

☐ `0xffffdc50`	☐ `0xffffdc20`	☐ `0xffffdbf0`
☐ `0xffffdc40`	☐ `0xffffdc10`	☐ `0xffffdbe0`
☐ `0xffffdc30`	☐ `0xffffdc00`	◯ None of the above

Q5 *Cryptography: "Sure, Whatever"* 🤷 **(9 points)**

Q5.1 (1 point) Alice sends (m, t) where $t = \text{MAC}(k, m)$. An attacker records one valid pair (m, t). Which attack on the protocol above is still possible even if the MAC is EUF-CMA secure?

- ◯ Replay the exact same (m, t) later and have Bob accept again.
- ◯ Modify m into m' and compute a valid t' for m'.
- ◯ Recover the MAC key k from (m, t).
- ◯ Distinguish $\text{MAC}(k, \cdot)$ from a random function using one query.

Q5.2 (1 point) Why does a perfectly secret one-time pad still fail to provide integrity?

- ◯ Because an attacker can flip bits in the ciphertext and predictably flip bits in the plaintext.
- ◯ Because OTP ciphertexts are distinguishable from random.
- ◯ Because OTP requires a PRG to be secure.
- ◯ Because OTP decryption is non-deterministic.

Q5.3 (1 point) What is the core risk in MAC-then-encrypt constructions that has enabled real attacks?

- ◯ The MAC key is sent in the clear.
- ◯ The MAC is computed on ciphertext rather than plaintext.
- ◯ The receiver decrypts attacker-controlled ciphertexts before checking integrity, leaking information via behavior/timing.
- ◯ The scheme cannot be CPA secure.

Q5.4 (1 point) In CPA security for public-key encryption, the adversary:

- ◯ Must not see the public key.
- ◯ Chooses two equal-length messages and receives an encryption of one of them under the public key.
- ◯ Gets a decryption oracle for arbitrary ciphertexts.
- ◯ Must guess the secret key.

Q5.5 (1 point) In "hashed ElGamal" (hybrid encryption), what is the ciphertext form for encrypting a message M to public key $\text{PK} = g^a$?

- ◯ $C = \left(g^{\text{ab}}, \text{AE.Enc}\left(H\left(g^b\right), M\right)\right)$.
- ◯ $C = \left(\text{AE.Enc}\left(H\left(g^{\text{ab}}\right), M\right)\right)$ with no group element sent.
- ◯ $C = (g^a, \text{AE.Enc}(H(g^a), M))$.
- ◯ $C = \left(g^b, \text{AE.Enc}\left(H\left((g^a)^b\right), M\right)\right)$ for fresh random b.

For Q5.6–Q5.7, consider a “hash-and-sign” signature scheme where:

- The signer computes $\sigma := \mathrm{Invert}(\mathrm{SK}, H(m))$.
- The verifier checks $F(\mathrm{PK}, \sigma) = H(m)$, where F is a trapdoor permutation.

Assume H maps messages into the domain X of the trapdoor permutation.

Q5.6 (1 point) Why do many security proofs for this scheme model H as a random oracle rather than assuming only that H is collision resistant?

- ◯ Collision resistance already implies pseudorandomness, so random oracle is redundant.
- ◯ The proof needs to “program” H on adversary queries in a way not captured by collision resistance/one-wayness alone.
- ◯ One-wayness of H is too strong and must be weakened to a random oracle assumption.
- ◯ Random oracle is required only to argue correctness, not security.

Q5.7 (1 point) Which fact best explains why an honestly generated signature on m will always be accepted by the verifier?

- ◯ H is collision resistant.
- ◯ H is one-way.
- ◯ $\mathrm{Invert}(\mathrm{SK}, \cdot)$ is the inverse of $F(\mathrm{PK}, \cdot)$ on the domain.
- ◯ The scheme is EUF-CMA secure.

Q5.8 (1 point) Which requirement is **not** part of the definition of a trapdoor one-way permutation?

- ◯ KeyGen outputs $(\mathrm{SK}, \mathrm{PK})$.
- ◯ $F(\mathrm{PK}, \cdot)$ is efficiently computable and is a permutation over a domain X.
- ◯ $\mathrm{Invert}(\mathrm{SK}, \cdot)$ efficiently computes the inverse of $F(\mathrm{PK}, \cdot)$.
- ◯ $F(\mathrm{PK}, \cdot)$ is collision resistant over X.

Q5.9 (1 point) Why is using a non-cryptographic PRNG to generate cryptographic keys dangerous?

- ◯ Non-cryptographic PRNGs are often too inefficient for repeated cryptographic use.
- ◯ Non-cryptographic PRNGs often leak enough data that observing enough outputs lets an attacker predict future outputs/state.
- ◯ Non-cryptographic PRNGs often fail enough statistical checks that attackers can directly recover hidden keys.
- ◯ Non-cryptographic PRNGs often produce only short outputs at a time.

Q6 *Cryptography: EvanCRHF* **(24 points)**

EvanBot is designing a hash function $\mathsf{H} : \{0,1\}^{2n} \to \{0,1\}^{2n}$.

Notation for the entire question:
- 0^n is an n-bit bitstring of all zeros.
- $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way, collision-resistant hash function (CRHF).

For each subpart with boxes, select whether the function is a CRHF.
- If "Yes," leave the boxes blank.
- If "No," give a collision pair $(x_1, x_2), (y_1, y_2)$ such that $\mathsf{H}(x_1, x_2) = \mathsf{H}(y_1, y_2)$ and $(x_1, x_2) \neq (y_1, y_2)$.

Example: If $\mathsf{H}(a, b) = a + b$, then a valid solution is $(x_1, x_2) = (2, 3),\ (y_1, y_2) = (4, 1)$.

Assumptions:
- x_1, x_2, y_1, y_2 are exactly n bits each, but you may answer with a simple integer and assume it is converted to the associated bitstring.
- There may be multiple correct answers. In the example above, $(x_1, x_2) = (1, 9),\ (y_1, y_2) = (5, 5)$ would also be correct.

Q6.1 (3 points) Is $\mathsf{H}(a, b) = a \| f(b)$ a CRHF?

○ Yes (leave the boxes blank) ○ No (fill in the boxes)

x_1 : x_2 :

y_1 : y_2 :

Q6.2 (3 points) Does $\mathsf{H}(a, b) = a \| f(b)$ possess one-wayness?

○ Yes, because an attacker must still invert $f(b)$ to find the complete pre-image (a, b).

○ Yes, because it is computationally difficult to determine where a ends and $f(b)$ begins.

○ No, because the first part of the input (a) is leaked directly in the output.

○ No, because the output length is strictly greater than the input length.

Q6.3 (3 points) Is $\mathsf{H}(a, b) = 0^n \| f(a)$ a CRHF?

◯ Yes (leave the boxes blank) ◯ No (fill in the boxes)

x_1 :

x_2 :

y_1 :

y_2 :

Q6.4 (3 points) Does $\mathsf{H}(a, b) = 0^n \| f(a)$ possess one-wayness?

◯ Yes, because fixing the first n bits prevents collisions across different inputs, which is enough to make it one-way.

◯ Yes, because an attacker still needs to invert $f(a)$ to find a valid a for the pre-image.

◯ No, because the construction leaks b directly in the output, so inverting is trivial by reading b.

◯ No, because the output only covers strings with prefix 0^n, so the function cannot be one-way for all outputs.

For the following **two subparts**, let the hash function from Q6.3 be G, so $G(a, b) = 0^n \| f(a)$.

Q6.5 (3 points) Is $\mathsf{H}(a, b) = G(G(a, b))$ a CRHF?

Note: The output of G is a 2n bit string. By splitting this in half, it is possible to produce a tuple of form (a,b) that can serve as the input to G.

◯ Yes (explain your answer below) ◯ No (explain your answer below)

Why or why not? Use 10 words or fewer (the staff answer is 5 words).

Q6.6 (3 points) Does $\mathsf{H}(a, b) = G(G(a, b))$ possess one-wayness?

◯ Yes, because a is not used to compute the output $\mathsf{H}(a, b)$.

◯ Yes, because b is not used to compute the output $\mathsf{H}(a, b)$.

◯ No, because the output is a constant for all inputs, making inversion trivial.

◯ No, because the output length is smaller than the input length, which always breaks one-wayness.

For the **remaining subparts**, EvanBot is attempting to design CRHF $\mathsf{H} : \{0,1\}^{2n} \to \{0,1\}^n$ that outputs n-bit strings.

Q6.7 (3 points) Is $\mathsf{H}(a, b) = a \oplus b \oplus f(a \oplus b)$ a CRHF?

○ Yes (leave the boxes blank) ○ No (fill in the boxes)

x_1 : x_2 :

y_1 : y_2 :

Q6.8 (3 points) Is $\mathsf{H}(a, b) = a^b \bmod p$ a CRHF?

Assume that a and b are cast to integers, and then the resulting integer $a^b \bmod p$ is cast back to an n-bit string.

○ Yes (leave the boxes blank) ○ No (fill in the boxes)

x_1 : x_2 :

y_1 : y_2 :

Q7 *Cryptography: Man Your Stations* (19 points)

Alice and Bob are trying to design their own key-exchange protocol.

Assume:

- Alice and Bob agree on a large prime p and generator g.
- They have signature key pairs (sk_A, pk_A) and (sk_B, pk_B), and both know each other's public keys.
- $\text{Enc}(K, M)$ is an IND-CCA secure encryption function with a corresponding decryption function.
- $\mathsf{H}(x)$ is a cryptographic hash function, e.g., SHA256.

The Protocol

1. Alice to Bob: Alice chooses a random x and computes $X = g^x \bmod p$. She sends X to Bob.

2. Bob to Alice: Bob chooses a random y and computes: $Y = g^y \bmod p$.
Bob derives the session key: $K = \mathsf{H}(\underline{\quad\text{Q7.1}\quad})$. He signs the transcript: $\sigma_B = \text{Sign}_{sk_B}(X \parallel Y)$.
Bob sends (Y, C_B) where $C_B = \text{Enc}(K, \sigma_B)$.

3. Alice to Bob: Alice computes: $K = \mathsf{H}(\underline{\quad\text{Q7.2}\quad})$. She decrypts C_B and verifies σ_B using pk_B.
If verification passes, Alice accepts K and she signs the transcript $\sigma_A = \text{Sign}_{sk_A}(X \parallel Y)$.
Alice sends $C_A = \text{Enc}(K, \sigma_A)$.

4. Final Step: Bob decrypts C_A and verifies σ_A using pk_A. If verification passes, Bob accepts K.

Q7.1 (2 points) Which of the following should fill in the blank in Step 2?

○ $X \bmod p$	○ $Y^x \bmod p$	○ $gXY \bmod p$
○ $Y \bmod p$	○ $X^y \bmod p$	○ $g \bmod p$
○ $XY \bmod p$	○ $g^{x+y} \bmod p$	○ $g^g \bmod p$

Q7.2 (2 points) Which of the following should fill in the blank in Step 3?

○ $X \bmod p$	○ $Y^x \bmod p$	○ $gXY \bmod p$
○ $Y \bmod p$	○ $X^y \bmod p$	○ $g \bmod p$
○ $XY \bmod p$	○ $g^{x+y} \bmod p$	○ $g^g \bmod p$

Q7.3 (2 points) Suppose Alice and Bob know authentic $(\text{pk}_A, \text{pk}_B)$ in advance. Why is the protocol secure against a man-in-the-middle attack by Mallory?

○ Because Enc is IND-CPA secure, Mallory cannot read C_A or C_B, so MITM is impossible.

○ Because each party signs the full transcript $(X \parallel Y)$ under an authentic public key, Mallory cannot alter X or Y (or swap in her own values) without causing signature verification to fail; she can relay messages but cannot make either party accept a key that Mallory knows.

○ Because H is collision resistant, Mallory cannot create a collision that would let her impersonate either party.

○ Because Diffie–Hellman alone is already secure against MITM, the signatures are unnecessary.

For Q7.4–Q7.6, Alice and Bob change their encryption protocol so that Bob no longer sends a signature to Bob in step 2. The steps of their modified protocol now look like:

1. Alice to Bob: Alice chooses a random x and computes $X = g^x \bmod p$. She sends X to Bob.

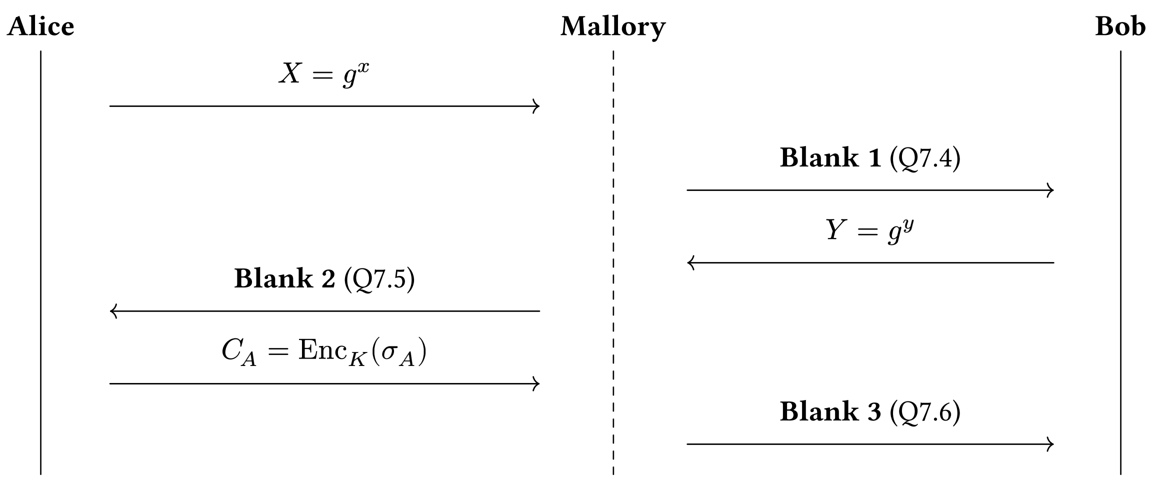
2. Bob to Alice: Bob chooses a random y and computes: $Y = g^y \bmod p$.
Bob derives the session key: $K = \mathsf{H}(\underline{\quad\text{Q7.1}\quad})$. **(No Signature!)**

3. Alice to Bob: Alice computes: $K = \mathsf{H}(\underline{\quad\text{Q7.2}\quad})$. She signs the transcript $\sigma_A = \text{Sign}_{sk_A}(X \parallel Y)$.
Alice sends $C_A = \text{Enc}(K, \sigma_A)$.

4. Final Step: Bob decrypts C_A and verifies σ_A using pk_A. If verification passes, Bob accepts K.

For Q7.4–Q7.6, Mallory is a MITM attacker between Alice and Bob, with two goals:
1. Mallory wants to trick Alice into forming a shared key between Alice and Mallory.
2. She also wants to trick Bob into forming a shared key between Bob and Mallory.

Mallory may or may not succeed in both goals, but she should succeed in at least one.



Clarification During Exam: The modified scheme contains a typo. It should read "Bob no longer sends a signature to ~~Bob~~ Alice in Step 2."

Clarification During Exam: The diagram should read $X = g^x \bmod p$ and $Y = g^y \bmod p$.

In the boxes below, you can use m to denote Mallory's secret. If a value can be anything, write "Anything."

Q7.4 (3 points) What should Mallory send to Bob in **Blank 1**?

Q7.5 (3 points) What should Mallory send to Alice in **Blank 2**?

Q7.6 (3 points) What should Mallory send to Bob in **Blank 3**?

Q7.7 (2 points) What shared K will Alice compute?

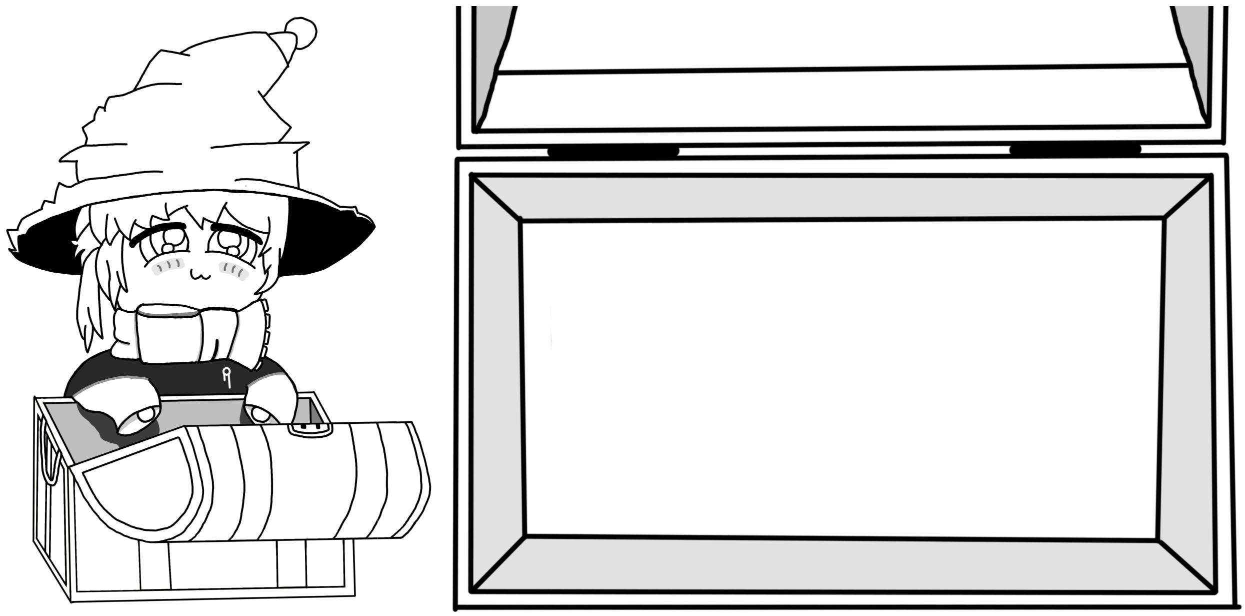
Q7.8 (2 points) After Mallory's attack, does Bob accept K?

○ Yes ○ No

If "Yes", write the value in the box. If "No", leave the box blank.

Post-Exam Activity: Beyond Bot's End

On a quest for magical grimoires, EvanBot sticks their head inside a chest in the depths of a dungeon. What's inside?



Art by: Soklynin Nou

Comment Box

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