CS 161 Spring 2024 — Introduction to Computer Security — Midterm

Name: _____

Student ID: _____

This exam is 110 minutes long.

Question:	1	2	3	4
Points:	0	16	16	16

Question:	5	6	7	Total
Points:	16	19	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.

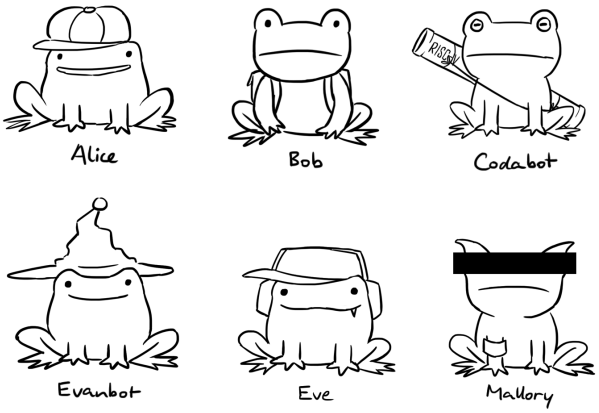
- ■ You can select
- ■ multiple squares (completely filled)

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 (0 points):
It's Leap Day, which means it's time for the quadrennial Leap-Frog Race! Everyone entered their frogs into the contest, but unfortunately the results got mixed up. EvanBot needs your help to piece together the standings from eyewitness reports!

1. Alice: "Bob's frog was right behind EvanBot's frog."
2. Bob: "CodaBot's frog was somewhere ahead of mine."
3. CodaBot: "I couldn't tell whose it was, but there was a frog with a hat somewhere ahead of Mallory's frog."
4. Eve: "My frog was always ahead of Alice's frog."
5. Mallory: "My frog definitely finished before CodaBot's frog. I'm telling the truth, I promise."

Circle the winning frog.



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: _____

Q2 *True/False* **(16 points)**

Each true/false is worth 1 point.

Q2.1 The Mallory Security Agency requires 128-bit AES encryption for all data, and "Top Secret" data is encrypted with an additional 256-bit key stored in a separate location.

True or False: This is an example of Defense in Depth.

○ (A) True ○ (B) False

Q2.2 Devices storing cryptographic keys are often required to be fitted with tamper-evident seals.

True or False: This is an example of Detect If You Can't Prevent.

○ (A) True ○ (B) False

Q2.3 True or False: All strings in C are terminated with the character `'\n'`.

○ (A) True ○ (B) False

Q2.4 True or False: Enabling non-executable pages prevents the ret2libc attack.

○ (A) True ○ (B) False

For the next 2 subparts: Suppose we have a little-endian C program with a local variable `char buf[8]`. Consider the following possible GDB output after running the command `x/4wx buf`:

```
0xffffff58: 0x61657270 0x0000006d 0xc0dab0bb 0xbaa15691
```

Q2.5 True or False: The words `0x61657270` and `0x0000006d` correspond to the data in `buf`.

○ (A) True ○ (B) False

Q2.6 True or False: `buf[9]` is `0xda`.

○ (A) True ○ (B) False

Q2.7 True or False: Every nondeterministic (randomized) encryption scheme is IND-CPA secure.

○ (A) True ○ (B) False

Q2.8 True or False: If Alice encrypts a message to Bob using ElGamal, an eavesdropper who can solve the discrete log problem would be able to correctly decrypt the message.

○ (A) True ○ (B) False

Q2.9 TRUE or FALSE: Digital signature schemes often encrypt the message before signing to prevent existential forgery attacks.

○ (A) TRUE ○ (B) FALSE

Q2.10 TRUE or FALSE: AES-CTR does not require the message to be padded.

○ (A) TRUE ○ (B) FALSE

Q2.11 TRUE or FALSE: HMAC is equivalent to NMAC with $K_1 = K_2$.

○ (A) TRUE ○ (B) FALSE

Q2.12 TRUE or FALSE: Using the output of a PRNG as the key for a one-time pad provides perfect security, even against an attacker with infinite computational power.

○ (A) TRUE ○ (B) FALSE

Q2.13 TRUE or FALSE: Length-extension attacks are often used to break the collision resistance of a hash function.

○ (A) TRUE ○ (B) FALSE

Q2.14 TRUE or FALSE: The security of Diffie-Hellman relies on the modulus being difficult to factor.

○ (A) TRUE ○ (B) FALSE

Q2.15 TRUE or FALSE: The security of ElGamal relies on the difficulty of finding $g^{ab} \bmod p$ given $g^a, g^b \bmod p$.

○ (A) TRUE ○ (B) FALSE

Q2.16 TRUE or FALSE: Certificates are generally not sent over insecure channels due to the risk of a replay attack.

○ (A) TRUE ○ (B) FALSE

Q3 *'Tis But a Scratch* (16 points)

King Arthur and his knights are searching for the Holy Grail. They find a castle protected by the following vulnerable C code:

```
1  void castle() {
2      int8_t holy_hand_grenade;
3      char[16] holy_grail;
4      char[128] cave;
5
6      // Implementation not shown.
7      find_grail(holy_grail);
8
9      memset(cave, 0, 128);
10
11     fread(&holy_hand_grenade, 1, 1, stdin);
12
13     if (holy_hand_grenade >= 128) {
14         return;
15     }
16
17     fread(cave, holy_hand_grenade, 1, stdin)
           ;
18     printf("%s", cave);
19 }
```

Stack at Line 8

RIP of `castle`
(1)
(2)
(3)
`cave`

The `find_grail` function writes a secret 16-byte string to `holy_grail`, and it's your job to make the program output the `holy_grail` string! You can assume `find_grail` does not modify the stack in any other way.

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

- ○ (A) (1) SFP of `castle` (2) `holy_hand_grenade` (3) `holy_grail`
- ○ (B) (1) SFP of `castle` (2) `holy_grail` (3) `cave`
- ○ (C) (1) RIP of `castle` (2) `holy_grail` (3) `holy_hand_grenade`
- ○ (D) (1) `&holy_grail` (2) RIP of `find_grail` (3) SFP of `find_grail`

Q3.2 (1 point) Which vulnerability is present in the code?

- ○ (A) Off-by-one
- ○ (B) Format string vulnerability
- ○ (C) Signed/unsigned vulnerability
- ○ (D) Heap overflow

In the next two subparts, provide inputs that would cause the program to output the `holy_grail` string (possibly as part of a larger output).

If a part of the input can be any non-zero value, use `'A'*n` to represent the `n` bytes of garbage.

Q3.3 (4 points) Input to `fread` at Line 11:

Q3.4 (4 points) Input to `fread` at Line 17:

The next four subparts are independent from each other.

Q3.5 (1 point) Would it still be possible for your exploit to leak `holy_grail` if `fread(&holy_hand_grenade, 1, 1, stdin)` on Line 11 is replaced with `gets(&holy_hand_grenade)`?

- ○ (A) Yes, because `gets` would allow the attacker to overwrite the address of `cave`.
- ○ (B) Yes, because `holy_hand_grenade` is above `holy_grail` in the stack diagram.
- ○ (C) No, the null terminator added by `gets` will cause the final `printf` to terminate before reaching `holy_grail`.
- ○ (D) No, because the null terminator added by `gets` will partially overwrite the SFP of `castle`.

Q3.6 (1 point) Would it still be possible for your exploit to leak `holy_grail` with stack canaries enabled?

- ○ (A) Yes, because the exploit writes around the canary to overwrite values above the canary.
- ○ (B) Yes, because the exploit never tries overwriting values above the canary.
- ○ (C) No, because all 4 bytes of the canary are overwritten by garbage.
- ○ (D) No, because the least-significant byte of the canary is overwritten by a null byte.

Q3.7 (1 point) Would it still be possible for your exploit to leak `holy_grail` with non-executable pages enabled?

- ○ (A) Yes, because the exploit never writes any executable instructions on the stack.
- ○ (B) Yes, because the malicious instructions being executed are not on the stack.
- ○ (C) No, because the exploit writes executable instructions on the stack.
- ○ (D) No, because non-executable pages stops the exploit from writing anything on the stack.

Q3.8 (1 point) Would it still be possible for your exploit to leak `holy_grail` with ASLR enabled?

- ○ (A) Yes, because the exploit does not require knowing any absolute addresses.
- ○ (B) Yes, because the `printf` call will always leak an address on the stack.
- ○ (C) No, because we need to know the address of `holy_grail` in order to leak its value.
- ○ (D) No, because we need to know the address of `castle` in order to complete the exploit.

For the following subparts, your goal is now to execute a 72-byte shellcode, rather than leaking `holy_grail`.

Q3.9 (1 point) Assuming no memory safety defenses are enabled, would it be possible to exploit this program to execute shellcode?

- ○ (A) Yes
- ○ (B) No

Q3.10 (1 point) Assuming canaries are enabled, would it still be possible to exploit this program to execute shellcode?

- ○ (A) Yes, because we can leak the canary and overwrite the canary with itself.
- ○ (B) Yes, because the `printf` call can write directly to the RIP, skipping the canary.
- ○ (C) No, because we run the final `fread` before we can leak the canary.
- ○ (D) No, because it is not possible to run shellcode at all in this program.

Q4 *I Sawed This Shellcode In Half!* **(16 points)**

Consider the following vulnerable C code:

```
1  void boat(void* shellcode_first_half, void* shellcode_second_half
       ) {
2      // fp contains the address of the fgets function
3      uintptr_t fp = (uintptr_t) fgets;
4
5      char[32] buf;
6      char* buf_ptr = &buf;
7
8      fgets(buf, 32, stdin);
9      printf(buf);
10
11     fgets(buf, 32, stdin);
12     printf(buf);
13 }
```

This is the result of running `disas fgets` in GDB:

```
1 0x08076030: push %ebp
2 0x08076034: mov %esp, %ebp
3 0x08076038: sub $20, %esp
4 ...
5 0x08076050: mov %ebp, %esp
6 0x08076054: pop %ebp
7 0x08076058: ret
```

`shellcode_first_half` is a pointer to the first half of shellcode and `shellcode_second_half` is a pointer to the second half of shellcode. Both halves of shellcode have a `ret` instruction at the end.

Assumptions:

- ASLR is enabled, but all other memory safety defenses are disabled.
- The program can print an arbitrarily large number of bytes using `printf`.
- All addresses are at least `0x01000000`.
- `esp` is not modified by the shellcode.

Q4.1 (2 points) Which of these inputs to the `fgets` on Line 8 will always leak the values of all the local variables in the `boat` stack frame? Select all that apply.

- ☐ (A) `"%x" * 10`
- ☐ (B) `"%s" * 10`
- ☐ (C) `"%n" * 10`
- ☐ (D) `"%c" * 10`
- ☐ (E) `"%d" * 10`
- ☐ (F) None of the above

Now that we have the values of all the local variables in the `boat` stack frame, we need to compute two values to use later in our exploit.

You can assume that the values outputted from the `printf` on Line 9 have all been converted to integers that you can perform arithmetic on.

Q4.2 (2 points) Which of these values do we need to use later in our exploit?

`&buf` is the address of `buf`, which you leaked in Q4.1.

Hint: What is the address in memory we want to write to?

- ○ (A) `&buf - 4`
- ○ (B) `&buf`
- ○ (C) `&buf + 32`
- ○ (D) `&buf + 36`
- ○ (E) `&buf + 40`
- ○ (F) `&buf + 44`

Q4.3 (2 points) Which of these values do we need to use later in our exploit?

`fp` is the address of the `fgets` function, which you leaked in Q4.1.

Hint: What is the value we want to write into memory?

- ○ (A) `fp`
- ○ (B) `fp + 0x4`
- ○ (C) `fp + 0x8`
- ○ (D) `fp + 0x20`
- ○ (E) `fp + 0x24`
- ○ (F) `fp + 0x28`

Let `x` be the value you computed in Q4.2, and `y` be the value you computed in Q4.3.

You can perform arithmetic on these values (e.g. `x + 5`), and you can assume that the resulting value is converted to the proper format (e.g. raw bytes, or decimal representation), so you don't have to manually do any conversions.

If a part of the input can be any non-zero value, use `'A'*n` to represent the `n` bytes of garbage.

Q4.4 (8 points) Provide an input to the `fgets` call on Line 11 that will execute shellcode.

[ ] + [ ] + `'%c' + '%` [ ] `u' + '%n'`

Q4.5 (1 point) If canaries are enabled, would it still be possible to exploit this program to execute shellcode?

- ○ (A) Yes, because the exploit writes directly to the RIP instead of smashing through the canary.
- ○ (B) Yes, because overwriting the RIP directly disables the canary check altogether.
- ○ (C) No, because all 4 bytes of the canary are overwritten by garbage.
- ○ (D) No, because the `ret` instruction at the end of the first shellcode will fail the canary check.

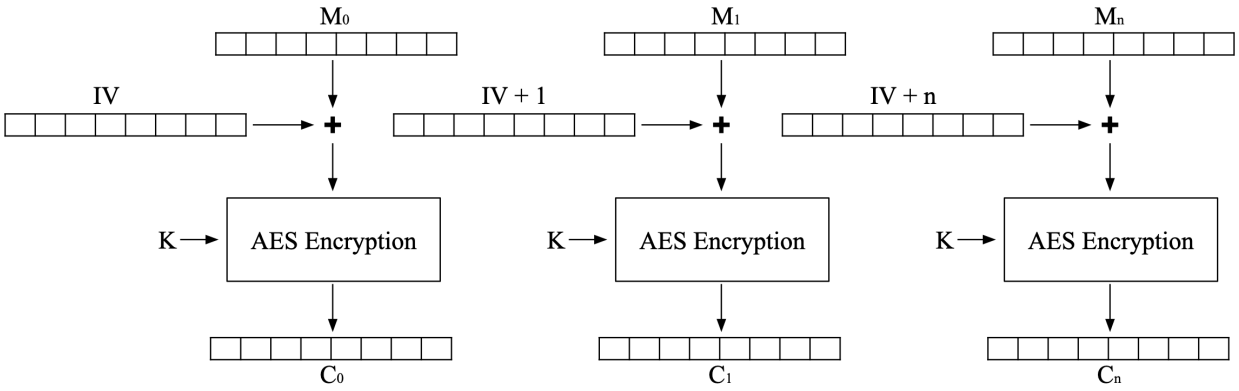
Q4.6 (1 point) What happens to the exploit if `boat` is called with the arguments reversed?

In other words, instead of `boat(shellcode_first_half, shellcode_second_half)`, what if we called `boat(shellcode_second_half, shellcode_first_half)`?

- ○ (A) The instructions of shellcode would all execute in reverse order.
- ○ (B) The shellcode would execute unchanged.
- ○ (C) The second half of shellcode executes, followed by the first half of shellcode.
- ○ (D) The program would crash after returning from `boat`, but before executing shellcode.

Q5 *Challenging Constructions* **(16 points)**

Consider the following encryption scheme:



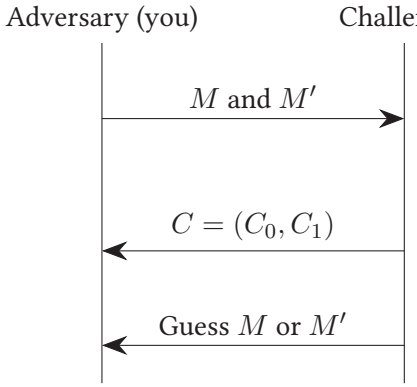
The encryption formula for this scheme is

$$C_i = E_K(M_i + IV + i)$$

Q5.1 (1 point) Select the correct decryption formula for the given scheme.

- ○ (A) $M_i = D_K(C_i - IV - i)$
- ○ (B) $M_i = D_K(C_i) - IV - i$
- ○ (C) $M_i = D_K(C_i) + IV + i$
- ○ (D) $M_i = D_K(C_i + IV - i)$

To show this scheme is insecure, you want to provide a strategy that always wins the IND-CPA game.



First, the adversary (that's you!) sends two different challenge messages, $M \neq M'$, to the challenger. For your strategy, you can assume M and M' are each two blocks long.

Then, the challenger randomly encrypts either M or M'. The resulting two-block ciphertext $C = (C_0, C_1)$ is returned to you.

Finally, you guess whether M or M' was encrypted.

In this strategy, the query phase is not needed (i.e. you never have to ask the challenger to encrypt messages of your choosing beforehand).

Assume that the second challenge message $M' = (?, ?)$ is chosen completely at random.

Clarification during exam: Assume that each "?" value is independently randomly generated, so (?, ?) would not necessarily be two identical values.

Q5.2 (2 points) What must be true of M for this strategy to work?

(Note: ? denotes a randomly-chosen value)

- ○ (A) $M_0 = 0$ and $M_1 = ?$
- ○ (B) $M_0 = ?$ and $M_1 = 0$
- ○ (C) $M_0 = ?$ and $M_1 = ?$
- ○ (D) $M_1 = M_0 + 1$
- ○ (E) $M_1 = M_0 - 1$
- ○ (F) $M_0 = M_1$

Q5.3 (3 points) The challenger encrypts one of M, M' from the previous subparts and returns $C = (C_0, C_1)$.

Explain how you would determine whether M or M' was encrypted. Your answer can use these values:

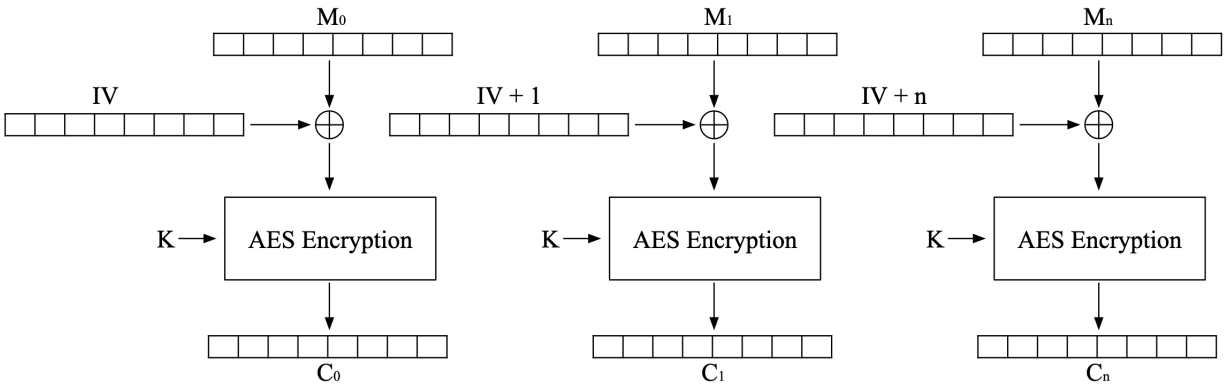
- $M = (M_0, M_1)$ and $M' = (M'_0, M'_1)$
- $C = (C_0, C_1)$. Recall: C is either the encryption of M or M'.

An example of how you could describe your strategy, that has nothing to do with this question: If $C_0 + 161 = M_0$, guess M. Else, guess M'.

Q5.4 (1 point) What is the probability (excluding negligible factors) that an optimal attacker (i.e. using the correct answer to Q5.3) wins the IND-CPA game?

- ○ (A) 50%
- ○ (B) 66.7%
- ○ (C) 75%
- ○ (D) 100%

Now consider a modified version of the previous encryption scheme:



The encryption formula for this scheme is

$$C_i = E_K(M_i \oplus (IV + i))$$

You want to devise a strategy to win the IND-CPA game.

Assume that the second challenge message $M' = (?, ?)$ is chosen completely at random.

Q5.5 (1 point) What value should you pick for M?

- ○ (A) $(0, ?)$
- ○ (B) $(?, 0)$
- ○ (C) $(?, ?)$
- ○ (D) $(0, 1)$
- ○ (E) $(1, 0)$
- ○ (F) $(1, 1)$

Q5.6 (5 points) The challenger encrypts one of M, M' from the previous subparts and returns $C = (C_0, C_1)$.

Explain how you would determine whether M or M' was encrypted. Your answer can use these values:

- $M = (M_0, M_1)$ and $M' = (M'_0, M'_1)$
- $C = (C_0, C_1)$. Recall: C is either the encryption of M or M'.

Q5.7 (3 points) What is the probability (excluding negligible factors) that an optimal attacker (i.e. using the correct answer to Q5.6) wins the IND-CPA game?

- ○ (A) 50%
- ○ (B) 62.5%
- ○ (C) 66.7%
- ○ (D) 75%
- ○ (E) 87.5%
- ○ (F) 100%

Justify your answer.

Q6 *Authentic Auctions* **(19 points)**

EvanBot wants to hold a charity auction and needs your help! They have designed a secure bidding protocol as follows:

1. Each party chooses a bid x.
2. Each party applies a function `COMMIT` to x and broadcasts $\texttt{COMMIT}(x)$ to every other party.
3. Once everyone has posted their commitment, each party reveals their bid and any other information relevant to the scheme.

There are two key properties we want to guarantee:

1. **Binding**: If a party chooses x, sends $\texttt{COMMIT}(x)$, and then reveals $x' \neq x$, other parties can detect that x' is invalid with overwhelming probability.
2. **Hiding**: $\texttt{COMMIT}(x)$ should not leak more than a negligible amount of information about x.

For each of the following schemes, select whether the scheme provides binding, hiding, both, or neither.

You should assume that the range of all possible bids is small enough to brute-force, each subpart is independent, and the auction is run exactly once. If a reveal step is not specified, assume that the reveal step is to send x.

Hint: EvanBot says you should reread the above assumptions and make sure you understand them before proceeding!

Clarification during exam: Assume that all parties have access to the trusted public keys of every other party.

Q6.1 (1 point) $\texttt{COMMIT}(x) = x$

○ (A) Binding ○ (B) Hiding ○ (C) Both ○ (D) Neither

Q6.2 (1 point) $\texttt{COMMIT}(x) = H(x)$.

○ (A) Binding ○ (B) Hiding ○ (C) Both ○ (D) Neither

Q6.3 (1 point) $\texttt{COMMIT}(x) = E_K(x)$, where K is a publicly-known symmetric key shared by everyone.

○ (A) Binding ○ (B) Hiding ○ (C) Both ○ (D) Neither

Q6.4 (1 point) $\texttt{COMMIT}(x) = x \oplus K$, where K is a randomly-generated one-time-pad key. To reveal, each party sends x, K.

○ (A) Binding ○ (B) Hiding ○ (C) Both ○ (D) Neither

Q6.5 (1 point) $\texttt{COMMIT}(x) = H(x) \oplus K$, where K is a randomly-generated one-time-pad key. To reveal, each party sends x, K.

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

The properties, repeated for your convenience:

1. **Binding**: If a party chooses x, sends $\texttt{COMMIT}(x)$, and then reveals $x' \neq x$, other parties can detect that x' is invalid with overwhelming probability.
2. **Hiding**: $\texttt{COMMIT}(x)$ should not leak more than a negligible amount of information about x.

Q6.6 (1 point) $\texttt{COMMIT}(x) = H(x \oplus K)$, where K is a randomly-generated one-time-pad key. To reveal, each party sends x, K.

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

Q6.7 (1 point) $\texttt{COMMIT}(x) = \mathsf{HMAC}(K, x)$, where K is a publicly-known symmetric key shared by everyone.

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

Q6.8 (1 point) $\texttt{COMMIT}(x) = \mathsf{Sign}(SK, x)$, where $\mathsf{Sign} = x^d \bmod N$ (naive RSA signature) and SK is the secret key for the user sending the bid.

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

Q6.9 (1 point) $\texttt{COMMIT}(x) = \mathsf{Sign}(SK, x)$, where $\mathsf{Sign} = H(x)^d \bmod N$ (hash-based RSA signature).

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

Q6.10 (1 point) $\texttt{COMMIT}(x) = g^x \bmod p$, where g is a generator and p is large prime (like in Diffie-Hellman).

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

Q6.11 (1 point) Note: (x, y) denotes a tuple of two values, x and y.

$$\texttt{COMMIT}(x) = \begin{cases} (x, x+1) & \text{with probability 0.5} \\ (x-1, x) & \text{with probability 0.5} \end{cases}$$

◯ (A) Binding ◯ (B) Hiding ◯ (C) Both ◯ (D) Neither

The properties, repeated for your convenience:

1. **Binding**: If a party chooses x, sends $\texttt{COMMIT}(x)$, and then reveals $x' \neq x$, other parties can detect that x' is invalid with overwhelming probability.
2. **Hiding**: $\texttt{COMMIT}(x)$ should not leak more than a negligible amount of information about x.

Q6.12 (8 points) Design a commitment scheme that was not mentioned in a previous subpart. Your scheme should be both binding and hiding.

Assume you have access to an unlimited source of random bits.

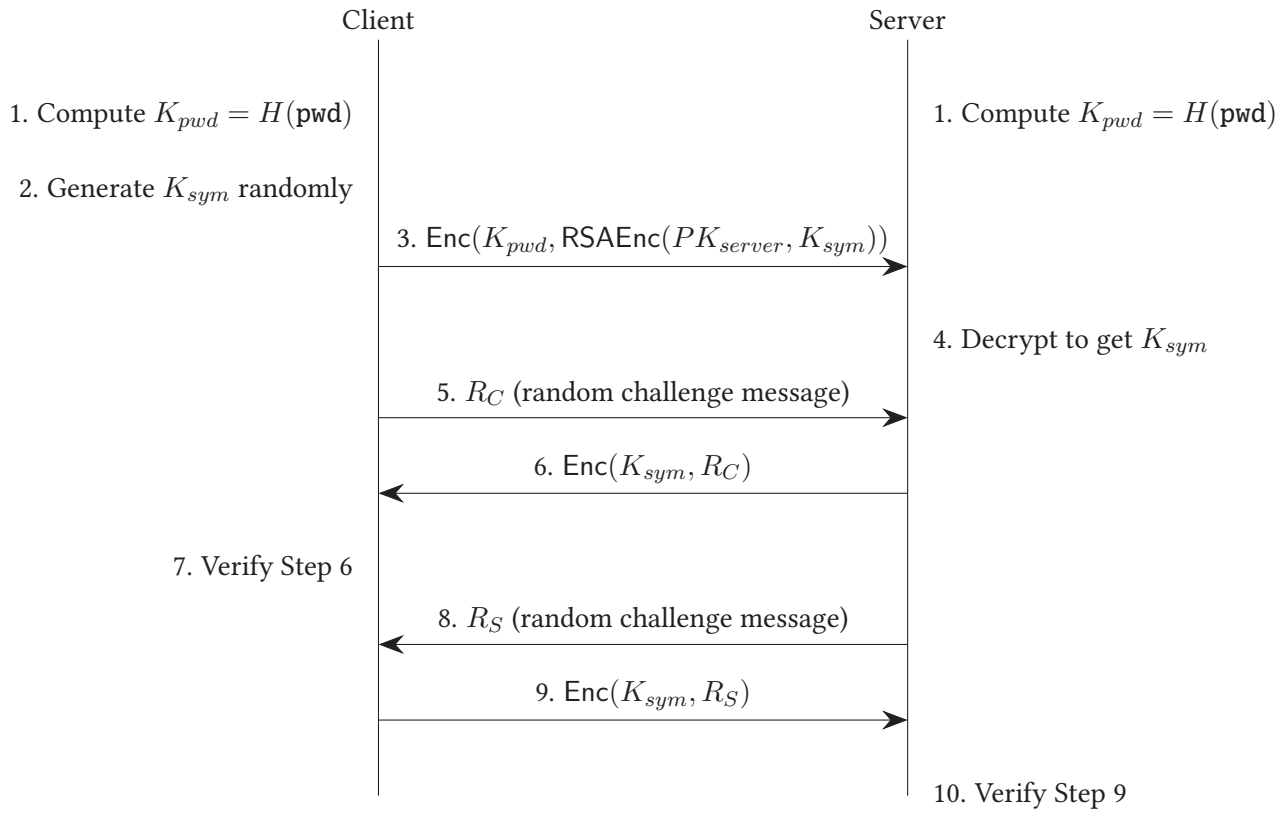
Given x, how do you compute $\texttt{COMMIT}(x)$?

List of values sent in the reveal step (including x):

List of steps to verify $\texttt{COMMIT}(x)$ given the values from the reveal step:

Q7 *Ephemeral Exchanges* **(17 points)**

Consider the following authentication protocol. `pwd` is a standard-strength password (i.e. vulnerable to brute-force). PK_{server} is a long-term, trusted public key for the server. Assume there's only a single user/password stored on the server.



Here is an equivalent description of the protocol:

1. Both the client and server derive $K_{pwd} = H(\texttt{pwd})$.
2. The client generates a random symmetric key K_{sym}.
3. The client sends $\mathsf{Enc}(K_{pwd}, \mathsf{RSAEnc}(PK_{server}, K_{sym}))$ to the server.
4. The server decrypts the message from the previous step to get K_{sym}.
5. The client sends a randomly generated number R_C to the server (challenge message).
6. The server replies with $\mathsf{Enc}(K_{sym}, R_C)$.
7. The client verifies that the server's response is valid.
8. The server sends a randomly generated number R_S to the client (challenge message).
9. The client replies with $\mathsf{Enc}(K_{sym}, R_S)$.
10. The server verifies that the client's response is valid.

Q7.1 (1 point) Which equation does the client use in Step 7 to verify the server's response from Step 6 (denoted STEP6)?

- ○ (A) $\mathsf{Dec}(K_{sym}, \texttt{STEP6}) = R_C$.
- ○ (C) $\mathsf{Dec}(K_{pwd}, \texttt{STEP6}) = R_C$.
- ○ (B) $\mathsf{Dec}(K_{sym}, \texttt{STEP6}) = R_S$.
- ○ (D) $\mathsf{Enc}(K_{sym}, \texttt{STEP6}) = R_C$.

Q7.2 (1 point) Which option best explains why the two parties don't use $H(\texttt{pwd})$ for the final shared symmetric key K_{sym}?

Clarification during exam (for 7.2, 7.3, 7.4): Assume that the attacker also records messages encrypted and MAC-d with K_{sym} after the protocol finishes.

- ○ (A) The attacker can brute force values of $H(\texttt{pwd})$ and check candidate keys using messages from the resulting secure channel.
- ○ (B) H outputs too many bits to be used as a symmetric key.
- ○ (C) $H(\texttt{pwd})$ would be the same across different authentications.
- ○ (D) $H(\texttt{pwd})$ would require a salt to prevent dictionary attacks.

Q7.3 (2 points) Is it possible for an eavesdropper to dictionary attack pwd given $\mathsf{Enc}(K_{pwd}, \mathsf{RSAEnc}(PK_{server}, K_{sym}))$ from Step 3?

- ○ (A) Yes, they can try decrypting $\mathsf{Enc}(K_{pwd}, \mathsf{RSAEnc}(PK_{server}, K_{sym}))$ with each guess of pwd.
- ○ (B) Yes, because anyone can evaluate $\mathsf{RSAEnc}(PK_{server}, K_{sym})$ and check that $\mathsf{Enc}(K_{pwd}, \mathsf{RSAEnc}(PK_{server}, K_{sym}))$ evaluates to the existing value.
- ○ (C) No, because there is no way to tell when a test decryption is successful.
- ○ (D) No, because RSAEnc is slow to evaluate, preventing brute force.

Q7.4 (2 points) An attacker records all messages sent between a client and server during a successful login.

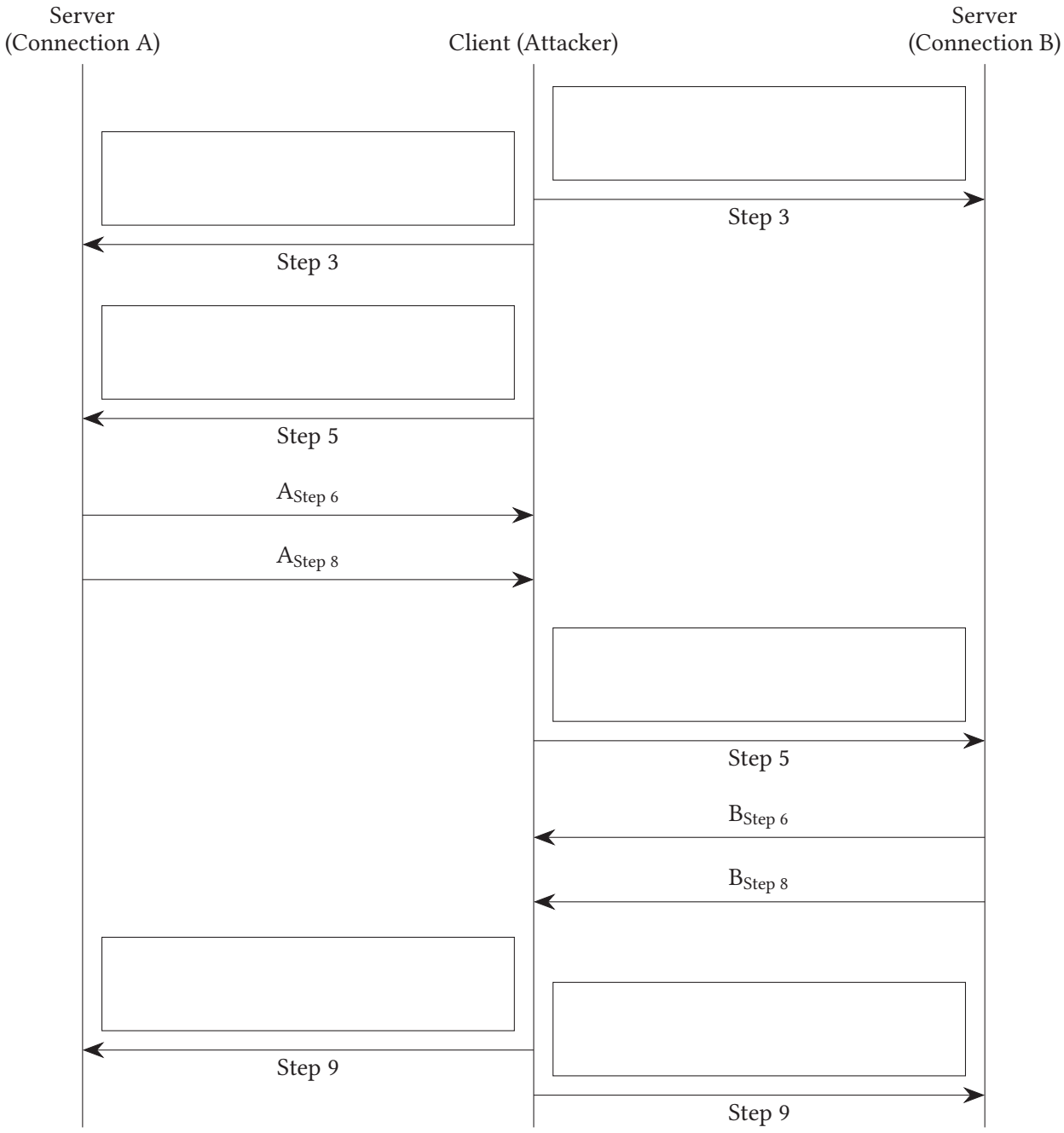
Later, the attacker learns the value of SK_{server}, the server's secret key (corresponding to PK_{server}).

Can the attacker learn the value of K_{sym} from the earlier recorded login?

- ○ (A) Yes, because the attacker can try all possible values of `pwd` and find K_{sym}.
- ○ (B) Yes, because the attacker can try all possible values of K_{sym} once SK_{server} is leaked.
- ○ (C) No, because K_{sym} is randomly generated for each authentication and deleted immediately afterwards.
- ○ (D) No, because there is no way to tell when a test decryption is successful, even if we leak SK_{server}.

Q7.5 (8 points) Design an attack to successfully pass Step 10 (server verification) without knowing `pwd`. Assumptions:

1. You start two simultaneous connections to the server: Connection A and Connection B. The same server is shown twice in the diagram below.
2. You do not need to verify the server's response in Step 7.
3. In each of the six boxes below, you can write:
   - $A_{\text{Step X}}$ or $B_{\text{Step X}}$ to denote the value from step X of Connection A/B.
   - If any value would work for a box, write the word `anything`.



Q7.6 (1 point) In which of the two connections does the server successfully verify the challenge response sent by the client in Step 9?

○ (A) Connection A ○ (B) Connection B ○ (C) Both connections

Q7.7 (2 points) Which of the following modifications would prevent an optimal attack from Q7.5? Select all that apply.

☐ (A) Adding an HMAC keyed with K_{sym} on the challenge messages (Steps 5, 8).

☐ (B) Having the server send its challenge message first (i.e. swap steps 5-7 with steps 8-10).

☐ (C) Replacing Step 6 with "The server replies with $\mathsf{Enc}(K_{sym}, R_C \| R_S)$"

☐ (D) Generating the challenge numbers using the output of a PRNG seeded with `pwd`.

☐ (E) None of the above

Nothing on this page will affect your grade.

Post-Exam Activity

What is Mallory cooking?

Comment Box

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