CS 161 Spring 2025	Introduction to Computer Security	Midterm

Solutions last updated: Mar 15, 2025

Name: ______________________________

Student ID: ____________________________

This exam is 110 minutes long.

Question:	1	2	3	4
Points:	0	12	18	20

Question:	5	6	7	Total
Points:	16	16	18	100

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

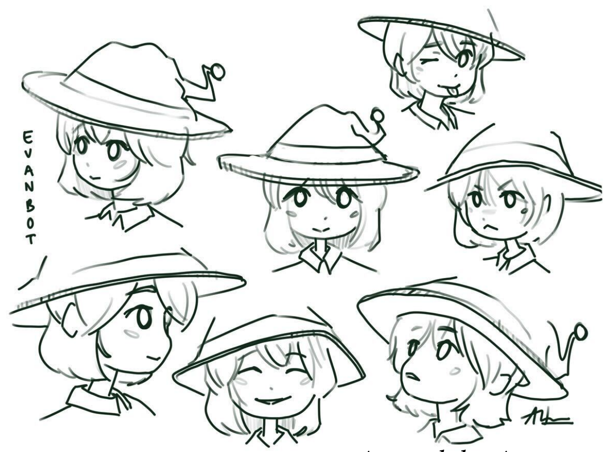
- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)
- Don't do this (it will be graded as incorrect)

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 may grade the worst interpretation.

Pre-exam activity (0 points):



Artwork by Anonymous

EvanBot here, EvanBot there, EvanBot everywhere!
Draw EvanBot from a different angle.

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* **(12 points)**

Each true/false is worth one point.

Q2.1 The Caltopia Space Agency only allows a few critical employees to control a space shuttle's flight path, while the rest of the employees only get enough access to carry out their work.

TRUE or FALSE: This is an example of Least Privilege.

● (A) TRUE ○ (B) FALSE

Q2.2 EvanBot designs a system that uses HMAC-DRBG with a truly random seed to generate secret keys used for symmetric encryption.

TRUE or FALSE: Based on Shannon's Maxim, we should assume that the attacker knows EvanBot is using HMAC-DRBG and can predict the generated secret key.

○ (A) TRUE ● (B) FALSE

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

```
0xfffd7014: 0xdeadbeef 0xffffffff 0xffff70ac 0x00000000
```

Q2.3 TRUE or FALSE: The value of `pancake[8]` is `0xff`.

○ (A) TRUE ● (B) FALSE

Q2.4 TRUE or FALSE: The value of `pancake[0]` is `0xef`.

● (A) TRUE ○ (B) FALSE

Q2.5 TRUE or FALSE: The first listed variable of a struct is stored at the lowest address.

● (A) TRUE ○ (B) FALSE

Q2.6 TRUE or FALSE: During a function call in x86, arguments are pushed onto the stack in the order they appear in the function definition.

○ (A) TRUE ● (B) FALSE

Q2.7 TRUE or FALSE: A buffer overflow vulnerability is impossible when stack canaries are enabled, because canaries protect the entire stack from arbitrary overwrites.

○ (A) TRUE ● (B) FALSE

Q2.8 TRUE or FALSE: CBC mode encryption is IND-CPA secure even if the IV is reused across multiple encryptions with the same key.

○ (A) TRUE ● (B) FALSE

Q2.9 TRUE or FALSE: RSA encryption without proper padding schemes (e.g., OAEP) is IND-CPA secure, provided the key size is sufficiently large.

○ (A) TRUE ● (B) FALSE

Q2.10 TRUE or FALSE: Rollback resistance ensures that an attacker cannot guess the next generated bit in a pseudorandom number generator.

○ (A) TRUE ● (B) FALSE

Q2.11 TRUE or FALSE: Public-key encryption is used in hybrid encryption because it can encrypt large amounts of data quickly.

○ (A) TRUE ● (B) FALSE

Q2.12 TRUE or FALSE: One-time pads are inconvenient because the keys can never be reused and need to be at least as long as the plaintext.

● (A) TRUE ○ (B) FALSE

Q3 *Pigeons In the Coal Mines - Memory Safety* **(18 points)**

Consider the following vulnerable C code:

```c
1  void foo() {
2      char buf[16];
3
4      fread(buf, 1, 16, stdin);
5      printf("%s", buf);
6      gets(buf);
7  }
8
9  int main() {
10     foo();
11     return 0;
12 }
```

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

Assumptions:

- Stack canaries are enabled, but no other memory safety defenses are enabled.
- You can use `SHELLCODE` as a 20-byte shellcode.
- You run GDB once and find that the address of `buf` is `0xffffffa0`.

Q3.1 (1 point) Fill the blanks in the stack diagram, assuming the program is paused on Line 3.

○ (A) (1) canary (2) `buf` (3) RIP of `foo`

● (B) (1) canary (2) RIP of `foo` (3) canary

○ (C) (1) RIP of `foo` (2) canary (3) canary

○ (D) (1) canary (2) RIP of `foo` (3) SFP of `foo`

Q3.2 (1 point) What type of vulnerability is present in this code?

○ (A) Format string vulnerability

● (B) Buffer overflow

○ (C) Signed/unsigned

○ (D) Off-by-one

In the next three subparts, provide an exploit that executes `SHELLCODE`.

Q3.3 (2 points) Give an input to `fread` on Line 4.

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

○ (A) `"A"*12 + "\xa0\xff\xff\xff"`

○ (B) `"A"*12 + "\xb8\xff\xff\xff"`

● (C) `"A"*16`

○ (D) `"A"*15 + "\x00"`

Q3.4 (2 points) Let `OUTPUT` be the value printed by the program from the `printf` on Line 5. Which slice of `OUTPUT` gives the value of the stack canary, assuming you have the correct input to the previous subpart?

Note: For example, `[0:4]` means the first four bytes of `OUTPUT`.

○ (A) `[0:4]`

○ (B) `[4:8]`

○ (C) `[8:12]`

○ (D) `[12:16]`

● (E) `[16:20]`

○ (F) `[20:24]`

Q3.5 (2 points) Let `CANARY` be the correct slice of `OUTPUT` from the previous subpart.

Give an input to `gets` on Line 6.

● (A) `"A"*16 + CANARY + "A"*4 + "\xbc\xff\xff\xff" + SHELLCODE`

○ (B) `"A"*16 + CANARY + "A"*4 + "\xb8\xff\xff\xff" + SHELLCODE`

○ (C) `SHELLCODE + CANARY + "\xa0\xff\xff\xff"`

○ (D) `SHELLCODE + "A"*4 + CANARY + "\xa0\xff\xff\xff"`

Solution: The goal of this exploit is to first leak the canary via `printf`, then use it in a stack smash to overwrite the RIP. Accordingly, our input to `fread` is `"A"*16`, which removes any potential null terminators between the start of `buf` and the canary itself. The canary is then printed by `printf("%s", buf)` in the `[16:20]` slice, since the canary is right after `buf` on the stack diagram.

Then our full exploit is 16 bytes of garbage, the canary to overwrite itself, another 4 bytes of garbage for the SFP, `&RIP + 4`, and then SHELLCODE.

Q3.6 (2 points) Which memory safety defenses, when enabled alongside stack canaries, would cause the correct exploit (without modifications) to fail? Consider each choice independently.

Note: For the PACs option only, assume the system is 64-bit (the exploit remains unchanged).

■ (A) Pointer authentication codes

■ (B) Non-executable pages

□ (C) None of the above

Solution: PACs: Disregarding any issues caused by the longest addresses, our exploit as-is uses RIP+4, which involves modifying a pointer.

NX pages: The exploit as-is executes SHELLCODE on the stack, and would therefore crash via NX pages.

For this rest of this question, **ASLR and stack canaries** are both enabled. In the next two subparts, provide an exploit that executes `SHELLCODE`.

Q3.7 (3 points) Give an input to `fread` on Line 4.

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

Solution: `"A"*16`

Q3.8 (5 points) Let `OUTPUT` be the output from the `printf` call on Line 5. You may slice this value (e.g. `OUTPUT[0:4]` returns the the first word of `buf`). You may also perform arithmetic on this value (e.g. `OUTPUT[0:4] - 7`) and assume it will be converted to/from the correct types automatically.

Also, let `CANARY` be the correct slice of `OUTPUT` from Q3.4.

Fill in each blank with an integer to provide an input to the `gets` call on Line 6.

Note that the `+` between terms refers to string concatenation (like in Project 1 syntax), but the minus sign in the second line refers to subtracting from the `OUTPUT[_:_]` value.

```
'A'*________ + CANARY + 'A'*________ +

(OUTPUT[______:______] - ________) + SHELLCODE
```

Solution: `"A"*16` + CANARY + "A"*4 + (OUTPUT[20:24]-4) + SHELLCODE

The key difference is the need to find a relative address to bypass ASLR. We can do this by using the SFP from the first printf, and modify it accordingly. The SFP of foo initially points to SFP of main, which is at RIP of foo + 8. We want to reach RIP of foo + 4, so we subtract 4 from this value.

Also possible (but not within the given skeleton) would be keeping OUTPUT[20:24] and adding 4 more garbage bytes at the end before putting SHELLCODE starting at SFP of main.

Q4 *ASLR (mod 5) - Memory Safety* **(20 points)**

Consider the following vulnerable C code:

```
1  void exploit() {
2      char buf[16];
3      size_t k = 0;
4
5      char new_byte = fgetc(stdin);
6      fgets(buf, 21, stdin);
7
8      size_t buflen = strlen(buf);
9      int n = 5;
10     while (n*k <= buflen) {
11         buf[n*k] = new_byte;
12         k += 1;
13     }
14 }
15
16 void sh_fn() {/* Code not shown */}
17
18 int main() {
19     // Function pointer
20     void (*shellcode)() = &sh_fn;
21     exploit();
22     return 0;
23 }
```

RIP of `main`
SFP of `main`
(1)
(2)
SFP of `exploit`
`buf`
(3)
`new_byte`
`buflen`
`n`

Non-executable pages are enabled. All other memory safety defenses are disabled.

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

```
0x080760A0: push %ebp
0x080760A4: mov %esp, %ebp
0x080760A8: sub $4, %esp
...
0x080760C8: call exploit
0x080760CC: mov $0, %eax
0x080760D0: add $4, %esp
0x080760D4: mov %ebp, %esp
0x080760D8: pop %ebp
0x080760DC: ret
```

Q4.1 (1 point) Fill in the blanks for the stack diagram, assuming the program is paused at Line 5.

- ○ (A) (1) `shellcode` (2) `buf` (3) RIP of `fgetc`
- ● (B) (1) `shellcode` (2) RIP of `exploit` (3) `k`
- ○ (C) (1) `shellcode` (2) RIP of `fgetc` (3) SFP of `fgetc`
- ○ (D) (1) RIP of `exploit` (2) `k` (3) RIP of `fgetc`

Q4.2 (2 points) What is the value of the RIP of `exploit`, assuming the program is paused on Line 5?

- ○ (A) `0x080760A4`
- ● (C) `0x080760CC`
- ○ (E) `0x080760D4`
- ○ (B) `0x080760C8`
- ○ (D) `0x080760D0`
- ○ (F) `0x080760DC`

Solution: The RIP of exploit points to the next instruction to execute once `exploit` returns, which is the one right after `call exploit`.

In the next two subparts, provide an exploit that causes the program to execute `sh_fn`.

Q4.3 (3 points) Provide an input to the `fgetc` on Line 5.

- ○ (A) `0x00`
- ○ (C) `0xA4`
- ○ (E) `0xD0`
- ○ (G) `0xD8`
- ○ (B) `0xA0`
- ○ (D) `0xA8`
- ○ (F) `0xD4`
- ● (H) `0xDC`

Q4.4 (3 points) Provide an input to the `fgets` on Line 6.

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

Solution: `A*20`

The goal of this exploit is to execute a ret2ret attack, by overwriting the RIP of exploit to equal `&ret`. This will then return into the shellcode function pointer stored right above the RIP of exploit on the stack.

The first key insight is that `&ret` is given in the assembly dump with LSB `0xDC`, and that this `&ret` shares the first three MSB with the existing RIP value of `0x080760CC`. Therefore, overwriting the LSB of RIP with `0xDC` is sufficient to change the overall pointer to `&ret`. This can be accomplished by setting `buf[20] = 0xDC`, since the RIP is exactly 20 bytes away from the start of buf.

However, the code as-is prevents us from writing anywhere `n*k <= buflen`, so we need to set `buflen` to be 20 or greater. We can do this by using the `fgets(buf, 21, stdin)` on Line 6 and writing 20 "A"s and 1 null terminator. It follows that `strlen(buf)` will return 20, enough for the while loop to execute `buf[20] = 0xDC` as intended. (Note that the fgets will initially overwrite the RIP LSB with a null terminator, but we overwrite this later on the while loop).

Q4.5 (3 points) How many different values of the variable `n` (on Line 9) (including `n = 5`) would result in the correct exploit, without modifications, executing `sh_fn`?

- ○ (A) 1
- ○ (B) 2
- ○ (C) 3
- ○ (D) 4
- ○ (E) 5
- ● (F) 6
- ○ (G) 7
- ○ (H) 8

Solution: We need to have `buf[20] = 0xDC`, so any value of `n` such that `n*k = 20` for some positive integer k. Therefore the answers are simply the divisors of 20: $1, 2, 4, 5, 10, 20$.

Q4.6 (2 points) Which memory safety defenses, when enabled alongside non-executable pages, would cause the correct exploit (without modifications) to fail? Consider each choice independently.

Note: For the PACs option only, assume the system is 64-bit (the exploit remains unchanged).

- ■ (A) Pointer authentication codes
- ■ (B) Stack canaries
- □ (C) None of the above

Solution: PACs will definitely break (regardless of issues involving larger pointer sizes) because we're expressly overwriting part of an existing pointer.

Stack canaries will also mess up the exploit by changing the size of the stack and distance from buf to RIP, making it such that we can't have high enough `buflen` to get `buf[20] = 0xDC`.

Q4.7 (3 points) Which modifications to the program itself would prevent the correct exploit, without modifications, from executing `sh_fn`?

Consider each choice independently.

- ■ (A) Changing Line 6 to `fgets(buf, 17, stdin)`
- ☐ (B) Changing Line 8 to `int buflen = strlen(buf)`
- ■ (C) Changing Line 10 to `while (n*k < buflen)`
- ☐ (D) Changing Line 12 to `k += 2`
- ☐ (E) None of the above

> **Solution:** Changing Line 6 to `fgets(buf, 17, stdin)`: This will prevent us from achieving any `buflen` higher than 16, preventing our exploit which requires running `buf[20] = 0xDC` (the while loop will terminate before then).
>
> Changing Line 8 to `int buflen = strlen(buf)` will not affect us, because the buflen value is positive and not large enough to overflow.
>
> Changing Line 10 to `while (n*k < buflen)` will prevent the exploit, because the maximum `buflen` size we can achieve is 20 due to the null terminator added by the fgets. Therefore when we reach `n*k = 20`, the while loop terminates instead of running `buf[20] = 0xDC`.
>
> Changing Line 12 to `k += 2` does not affect us with the given `n=5`, since `k=4` will give `n*k = 20` as needed.

Q4.8 (3 points) In this subpart only, **assume ASLR is also enabled.** What is the approximate probability that the correct exploit, without modifications, executes `sh_fn`?

Clarification after exam: Assume ASLR randomizes the code section.

○ (A) 0 ● (B) $\frac{1}{256}$ ○ (C) $\frac{1}{2}$ ● (D) 1

> **Solution:** 1 was given credit because the question did not specify whether the code section was randomized, even though this was the intended setup.
>
> ASLR randomizing the code section is an issue for our exploit, since we don't have a fixed byte like `0xDC` to overwrite the RIP LSB with. However, we know that a `ret` is always 16 bytes ahead of the current value in `RIP of exploit` because ASLR maintains relative addressing.
>
> Therefore, our best try is to overwrite with some value greater than `0x10` and hope that this hits the `ret`. This (very roughly) can be expected to occur with probability $\frac{1}{256}$ – certainly this is the best option as it isn't impossible (probability 0) or the other two options which have probability way higher than expected.

Q5 *AES-COMBO - Symmetric Cryptography* **(16 points)**

EvanBot designs the AES-COMBO mode of operation, defined below:

$$C_1 = E_K(IV_1 \oplus P_1)$$
$$C_2 = E_K(IV_2 \oplus P_2) \oplus C_1$$
$$C_i = E_K(C_{i-2} \oplus P_i)$$

Q5.1 (1 point) Select the correct decryption formula for $i \geq 3$.

- ○ (A) $P_i = D_K(C_i \oplus C_{i-2})$
- ○ (C) $P_i = D_K(C_i) \oplus C_{i-1}$
- ○ (B) $P_i = E_K(C_i) \oplus C_{i-1}$
- ● (D) $P_i = D_K(C_i) \oplus C_{i-2}$

Q5.2 (3 points) Select all methods for generating IV_1 and IV_2 that result in AES-COMBO being IND-CPA secure.

All choices are independent of each other.

- ■ (A) IV_1 and IV_2 are independently randomly generated.
- ☐ (B) Seed a PRNG with K, set $IV_1 =$ `Generate(128)`, and then set $IV_2 =$ `Generate(128)` using the same PRNG instance.
- ☐ (C) Seed two separate PRNGs with K, set $IV_1 =$ `Generate(128)` from the first PRNG, and then set $IV_2 =$ `Generate(128)` from the second PRNG.
- ■ (D) IV_1 is randomly generated and $IV_2 = H(IV_1)$.
- ■ (E) IV_2 is randomly generated and $IV_1 = H(IV_2)$.
- ☐ (F) None of the above

Solution:

- **IV_1 and IV_2 are independently randomly generated:**

  This case essentially reduces to two chains of AES-CBC: one for odd blocks and one for even blocks. The $\oplus C_1$ factor in C_2 does not affect anything, as its an XOR with a known value.

- **Seed a PRNG with K, set $IV_1 =$ `Generate(128)`, and then set $IV_2 =$ `Generate(128)` using the same PRNG instance.**

  The PRNG seeding with K makes it deterministic for the purposes of IND-CPA.

- **Seed two separate PRNGs with K, set $IV_1 =$ `Generate(128)` from the first PRNG, and then set $IV_2 =$ `Generate(128)` from the second PRNG.**

  The key here is that this results in $IV_1 = IV_2$. We then can break IND-CPA as given in the latter half of this question in Q5.5/5.6.

- **IV_1 is randomly generated and $IV_2 = H(IV_1)$:**

  Given that IV_1 is random, $H(IV_2)$ also appears pseudorandom and unpredictable for the attacker. Therefore the scheme is essentially equivalent to Option A.

- **IV_1 is randomly generated and $IV_2 = H(IV_1)$:**

  Given that IV_2 is random, $H(IV_1)$ also appears pseudorandom and unpredictable for the attacker. Therefore the scheme is essentially equivalent to Option A.

In the next two subparts, suppose a ciphertext C gets modified in transit. Let P' represent the plaintext from decrypting C'. For each row, select the corresponding value. "Garbage" refers to a pseudorandom string, e.g. an unknown value decrypted with a block cipher.

Q5.3 (3 points) C is modified such that $C_1' = C_1 \oplus 1$ (i.e. a bit flip in the first ciphertext block).

P_1':	● (A) Garbage	○ (B) $P_1 \oplus 1$	○ (C) $P_1 \oplus P_2$	○ (D) P_1
P_2':	● (A) Garbage	○ (B) $P_2 \oplus 1$	○ (C) $P_2 \oplus P_1$	○ (D) P_2
P_i', $i \geq 5$, i **even**:	○ (A) Garbage	○ (B) $P_i \oplus 1$	○ (C) $P_i \oplus P_{i-1}$	● (D) P_i
P_i', $i \geq 5$, i **odd**:	○ (A) Garbage	○ (B) $P_i \oplus 1$	○ (C) $P_i \oplus P_{i-1}$	● (D) P_i

Solution: Recall the decryption formulas:

$$P_1 = D_K(C_1) \oplus IV_1$$
$$P_2 = D_K(C_2 \oplus C_1) \oplus IV_2$$
$$P_i = D_K(C_i) \oplus C_{i-2}$$

Plug in $C_1' = C_1 \oplus 1$ (and everything else remains the same):

$$P_1' = D_K(C_1') \oplus IV_1'$$
$$P_1' = D_K(C_1 \oplus 1) \oplus IV_1$$
$$P_1' = \text{garbage} \oplus IV_1$$
$$P_1' = \text{garbage}$$

(Remember "garbage" is just a pseudorandom-looking string, for instance decrypting an unknown value with a block cipher as we did here)

$$P_2' = D_K(C_2' \oplus C_1') \oplus IV_2'$$
$$P_2' = D_K(C_2 \oplus C_1 \oplus 1) \oplus IV_2$$
$$P_2' = D_K(E_K(IV_2 \oplus P_2) \oplus 1) \oplus IV_2$$
$$P_2' = \text{garbage} \oplus IV_2$$
$$P_2' = \text{garbage}$$

Since $i \geq 5$, the last two options are unaffected, as they only reference C_3 or later, and only C_1' was modified.

$$P_i' = D_K(C_i') \oplus C_{i-2}'$$
$$P_i' = D_K(C_i) \oplus C_{i-2}$$
$$P_i' = P_i$$

Q5.4 (3 points) C is modified such that $C_2' = C_2 \oplus 1$.

P_1':	○ (A) Garbage	○ (B) $P_1 \oplus 1$	○ (C) $P_1 \oplus P_2$	● (D) P_1
P_2':	● (A) Garbage	○ (B) $P_2 \oplus 1$	○ (C) $P_2 \oplus P_1$	○ (D) P_2
P_i', $i \geq 5$, i **even**:	○ (A) Garbage	○ (B) $P_i \oplus 1$	○ (C) $P_i \oplus P_{i-1}$	● (D) P_i
P_i', $i \geq 5$, i **odd**:	○ (A) Garbage	○ (B) $P_i \oplus 1$	○ (C) $P_i \oplus P_{i-1}$	● (D) P_i

Solution: Plug in $C_2' = C_2 \oplus 1$ (and everything else remains the same):

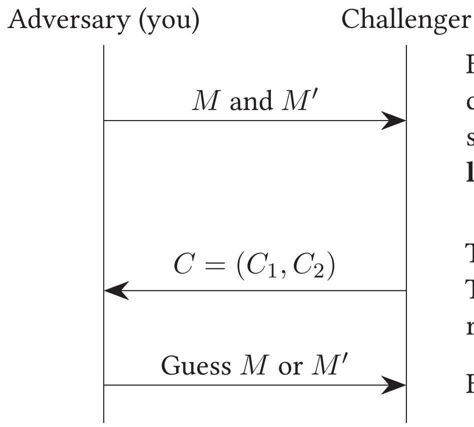
$$P_1' = D_K(C_1') \oplus IV_1'$$
$$P_1' = D_K(C_1) \oplus IV_1$$
$$P_1' = P_1$$

$$P_2' = D_K(C_2' \oplus C_1') \oplus IV_2'$$
$$P_2' = D_K(C_2 \oplus 1 \oplus C_1) \oplus IV_2$$
$$P_2' = D_K(E_K(IV_2 \oplus P_2) \oplus 1) \oplus IV_2$$
$$P_2' = \text{garbage} \oplus IV_2$$
$$P_2' = \text{garbage}$$

Since $i \geq 5$, the last two options are unaffected, as they only reference C_3 or later, and only C_1' was modified.

$$P_i' = D_K(C_i') \oplus C_{i-2}'$$
$$P_i' = D_K(C_i) \oplus C_{i-2}$$
$$P_i' = P_i$$

Assume for the following subparts only that $\mathbf{IV_1} = \mathbf{IV_2} = \mathbf{IV}$ for some randomly generated IV. You want to provide a strategy to win 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 $C = (IV, C_1, C_2)$ is returned to you.

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

NOTE: The diagram originally had a typo with $C = (C_0, C_1)$.

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).

The second challenge message $M' = (?, ?)$ is **two randomly-generated blocks**.

Q5.5 (2 points) What must be true of $M = (M_1, M_2)$ for this strategy to work?

Note: $?$ denotes a randomly-chosen value.

○ (A) $M_1 = 0$ and $M_2 = ?$

○ (B) $M_1 = ?$ and $M_2 = 0$

○ (C) $M_1 = ?$ and $M_2 = ?$

○ (D) $M_1 \neq M_2$

○ (E) $M_1 = M_2 \oplus 1$

● (F) $M_1 = M_2$

Q5.6 (4 points) Assume that M satisfies the condition you gave for the previous subpart.

Let $C = (IV, C_1, C_2)$ be the challenge ciphertext. Provide a strategy to guess whether M or M' was picked, with non-negligibly higher than 50% probability.

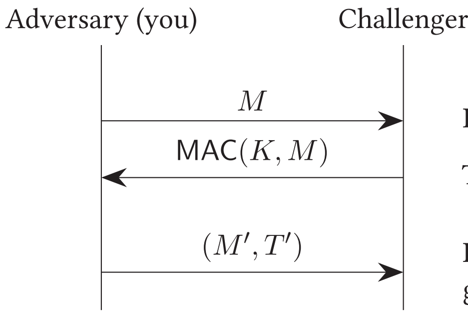
Your answer should be formatted along the lines of "If $C_1 \oplus 161 = 0$, then guess M, else guess M'" (no relation to actual solution).

Solution: If $C_2 = 0$, output M, otherwise M'.

Our strategy uses the fact that, when $IV_1 = IV_2$, we have $C_1 = E_K(IV \oplus M_1)$ and $C_2 = E_K(IV \oplus M_2) \oplus C_1$ (note we switched to M_i instead of P_i here due to IND-CPA convention, but the formulas are equivalent) . When $M_1 = M_2 = X$, we have $C_1 = E_K(IV \oplus X)$ and $C_2 = E_K(IV \oplus X) \oplus C_1 = E_K(IV \oplus X) \oplus E_K(IV \oplus X) = 0$.

Q6 *A Song of MACs and Signatures - Cryptography* **(16 points)**

EvanBot wants to review alternatives to HMACs and signatures. Below is a simplified version of the EU-CMA (referred to as EU-CPA in lecture) security game, with **only 1 query message** M (which will be sufficient for all subparts).



First, you send one query message M to the challenger.

Then, the challenger returns the MAC of M.

Finally, you output some $M' \neq M$ and a tag T', and win the game if T' is a valid MAC for M' with non-negligible probability.

In each subpart, select whether the given scheme is EU-CMA secure. If you selected "Insecure", provide an attack to win the EU-CMA game with non-negligible probability. If you selected "Secure", leave the boxes blank.

For all subparts: **if a box can be an arbitrary value, you must put "anything"** as the answer.

Q6.1 (4 points) $\mathsf{MAC}(K, M) = \mathsf{CBC}(K, H(M)) = (IV, C)$

CBC is AES-CBC encryption. IV is randomly generated per MAC. H has an output of 128 bits.

○ (A) Secure ● (B) Insecure

Query Message: **Solution:** anything

Let $T = (IV, C)$ be the tag received for the query message M. Now provide a pair (M', T') such that $M' \neq M$ and T' is a valid MAC for M' with non-negligible probability.

Solution Message: **Solution:** anything

Note: Recall that the tag in this scheme is a pair of the form (IV, C).

Solution Tag:

Solution: $(IV \oplus H(M') \oplus H(M), C)$.
When verifying the MAC (IV', C') as given above, the user runs CBC with IV as given in the tag (required to deterministically verify), such that

$$\begin{aligned}
\mathsf{CBC}(K, H(M')) \text{ using } IV' &= E_K(IV' \oplus H(M')) \\
&= E_K(IV \oplus H(M') \oplus H(M) \oplus H(M')) \\
&= E_K(IV \oplus H(M)) \\
&= C = C'
\end{aligned}$$

therefore our (IV', C') verifies correctly for a new **arbitrary** message M'.

Remember: **if a box can be an arbitrary value, you must put "anything"** as the answer.

Q6.2 (4 points) $\mathsf{MAC}(K, M) = \mathsf{CTR}(K, H(M)) = (IV, C)$

CTR is AES-CTR encryption. IV is randomly generated per MAC. H has an output of 128 bits.

○ (A) Secure

● (B) Insecure

Query Message:

Solution: anything

Let $T = (IV, C)$ be the tag received for the query message M. Now provide a pair (M', T') such that $M' \neq M$ and T' is a valid MAC for M' with non-negligible probability.

Solution Message:

Solution: anything

Note: Recall that the tag in this scheme is a pair of the form (IV, C).

Solution Tag:

Solution: $(IV, C \oplus H(M) \oplus H(M'))$
When verifying the MAC (IV', C') as given above:

$$\begin{aligned} C' &= E_K(IV) \oplus H(M') \oplus H(M) \oplus H(M') \\ &= E_K(IV) \oplus H(M') \end{aligned}$$

therefore our (IV', C') verifies correctly for a new **arbitrary** message M'.

For each of the following signature schemes, answer whether the scheme is EU-CMA secure.

The EU-CMA game for signature schemes is identical to the game for MACs, except the challenger returns the signature of the query message under the secret key SK for their public key PK. Your goal as the adversary is to output a valid message/signature pair (M', S') for PK with M' different from the query message.

Q6.3 (4 points) $\mathsf{Sign}(SK, M) = M^d \bmod N$

$d = SK$ is an RSA private key associated with the public key (e, N).

M must satisfy $2 \leq M \leq N - 2$.

○ (A) Secure ● (B) Insecure

Query Message:

Solution: anything

Let S be the signature received for the query message M. Now provide a pair (M', S') such that $M' \neq M$ and S' is a valid signature for M' with non-negligible probability.

Solution Message:

Solution: M^2

Solution Signature:

Solution: S^2
This is one of many possible answers – you could also do existential forgery e.g. S = anything and then $M = S^e$. However it's not possible to have M = anything.

Q6.4 (4 points) $\mathsf{Sign}(SK, M) = H(M) + xM \bmod p$

$x = SK$ is the private key chosen uniformly at random $\bmod p$, with the public key $PK = g^x$.

M must satisfy $2 \leq M \leq p - 2$.

$\mathsf{Verify}(PK, (S_1, S_2))$ returns `true` if $g^{-H(M)} \cdot g^S = (PK)^M \bmod p$.

Clarification after exam: $\mathsf{Verify}(PK, (S_1, S_2))$ *should read* $\mathsf{Verify}(PK, S)$.

○ (A) Secure ● (B) Insecure

Query Message:

Solution: anything

Let S be the signature received for the query message M. Now provide a pair (M', S') such that $M' \neq M$ and S' is a valid signature for M' with non-negligible probability.

Solution Message:

Solution: anything

Solution Signature:

Solution:
While it's possible to break this scheme algebraically, it's easiest is to solve for x given the signature, then sign normally. Note that $S = H(M) + xM \bmod p$, and that the adversary knows M by construction, so they can find $x \equiv (S - H(M)) \cdot M^{-1} \bmod p$.

Then we can use the sign formula: $S' = \mathsf{Sign}(x, M') = H(M') + x = H(M') + ((S - H(M)) \cdot M^{-1} \bmod p - 1$.

Alternative solution that doesn't allow for arbitrary solution message: Arbitrary query, solution message = $2M$, solution signature: $2(S - H(M)) + H(M')$.

This page intentionally left (mostly) blank.

The exam continues on the next page.

Q7 *Be My Proxy? - Asymmetric Cryptography* (18 points)

Consider the following variant of ElGamal encryption. For all of Q7, assume that H outputs 128 bits.

Key Generation:

1. Choose a random private key $b \bmod p$ such that $\gcd(b, p-1) = 1$.
2. Derive the public key as $B = g^b \bmod p$.

Encryption:

1. Choose a random $r \bmod p$ such that $\gcd(r, p-1) = 1$.
2. Compute $R = g^r \bmod p$.
3. Let $K = H(B^r \bmod p)$ (i.e. the hash of $B^r \bmod p$).
4. Send $(C_1, C_2) = (R, \mathsf{Enc}(K, M))$.

Decryption:

1. Compute $K = H(\_\_\_\_\_\_\_\_)$.
2. Decrypt $M = \mathsf{Dec}(K, C_2)$.

Q7.1 (1 point) What goes in the blank in the decryption protocol?

● (A) $C_1^b \bmod p$ ○ (B) $C_1^B \bmod p$ ○ (C) $B^{C_1} \bmod p$ ○ (D) $B^r \bmod p$

> **Solution:** $C_1^b \equiv (g^r)^b \equiv g^{br} \equiv B^r \bmod p$

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

■ (A) The variant scheme is IND-CPA secure.

☐ (B) The variant scheme is multiplicatively malleable (e.g. a ciphertext C encrypting M can be transformed into a ciphertext C' encrypting $2M$, without knowing b).

☐ (C) The variant scheme is additively malleable (e.g. a ciphertext C encrypting M can be transformed into a ciphertext C' encrypting $M + 1$, without knowing b).

☐ (D) None of the above

> **Solution:** This scheme is essentially ElGamal from lecture but instead of doing $M \cdot K \bmod p$, we use K for symmetric encryption.
>
> This use of symmetric encryption prevents any form of malleability.

Q7.3 (2 points) Recall that the ElGamal scheme from lecture defines $C_2 = M \cdot B^r \bmod p$ instead of $\mathsf{Enc}(H(B^r \bmod p), M)$.

Alice and Bob believe that this variant scheme will protect them against a man-in-the-middle attack from Mallory, unlike lecture ElGamal. Assume that Alice and Bob do **not** know each other's public keys and must first share them over the insecure channel.

Is this correct?

- ○ (A) Yes, because Mallory can't predictably modify C_2.
- ○ (B) Yes, because $M \cdot B^r \bmod p$ is not confidential (i.e. it leaks some information about M).
- ○ (C) No, because Enc only provides authenticity if the attacker doesn't know the key.
- ● (D) No, because Mallory can still cause Alice and Bob to derive keys known to Mallory.

Solution: Since Alice and Bob have to share their public keys first, Mallory can easily intercept and replace these with keys known to her. Then she can MITM attack the scheme.

Q7.4 (3 points) The hardness of which cryptographic problems is necessary for the variant scheme to be secure? Select all that apply.

- ■ (A) Discrete logarithm problem
- ■ (B) Diffie-Hellman problem
- □ (C) RSA problem
- □ (D) None of the above

Solution: An adversary who can break DLP can break the DHP.

Given the ability to break DHP, we can recover $g^{br} \bmod p$ from $C_1 = g^r \bmod p$ and $B = g^b \bmod p$, from which we can derive K.

The RSA problem is not applicable here.

Alice is about to leave on a month-long vacation, and wants the central mail server at her office to redirect all her email to Bob's inbox. However, since she uses encrypted email, Bob won't be able to read these messages as they were encrypted with B_{Alice} (Alice's public key).

They decide to use this ElGamal variant to develop a **proxy re-encryption** system. This system allows transforming ciphertexts encrypted with B_{Alice} to be encrypted with B_{Bob} instead, while keeping the underlying plaintext the same.

Q7.5 (6 points) Design a proxy re-encryption protocol using the modified ElGamal scheme. That is, design an algorithm to transform $C = (C_1, C_2) = (g^r \bmod p, \mathsf{Enc}(H(B^r_{\text{Alice}} \bmod p), M)$ encrypting some message M into $C' = (C'_1, C'_2)$ that decrypts to the same message M when decrypted by Bob with b_{Bob}.

Clarification after exam: *The original subpart had a typo, saying* $C_2 = \mathsf{Enc}(K, H(B^r_{Alice} \bmod p))$ *instead of the correct* $\mathsf{Enc}(H(B^r_{Alice} \bmod p), M)$ *as given in the protocol.*

First, the mail server is given a specific value V that will enable proxy re-encryption.

V:

- ● (A) $b_{\text{Alice}} \cdot b^{-1}_{\text{Bob}} \bmod (p-1)$
- ○ (B) $b_{\text{Bob}} \cdot b^{-1}_{\text{Alice}} \bmod (p-1)$
- ○ (C) $b_{\text{Bob}} \cdot b_{\text{Alice}} \bmod (p-1)$
- ○ (D) $b_{\text{Bob}} + b_{\text{Alice}} \bmod (p-1)$

Given $C = (C_1, C_2)$ and V, give an expression for $C' = (C'_1, C'_2)$:

C'_1:

- ○ (A) C_1
- ○ (B) $C_1 + V \bmod (p-1)$
- ○ (C) $C_1 \cdot V \bmod p$
- ● (D) $C_1^V \bmod p$

C'_2:

- ● (A) C_2
- ○ (B) $C_2 + V \bmod (p-1)$
- ○ (C) $C_2 \cdot V \bmod p$
- ○ (D) $C_2^V \bmod p$

Solution: Note that the mail server has no hope of changing the value of K, since any change to C_2 will cause it to decrypt garbage and the server cannot learn the value of M directly. Therefore we must cause Bob's decryption process to derive the same K that was used to create C_2.

However, Bob will derive $K' = (g^r)^{b_{\text{Bob}}}$ which does not equal $K = (g^r)^{b_{\text{Alice}}}$. Accordingly we need to change C_1 such that $(C'_1)^{b_{\text{Bob}}} = (g^r)^{b_{\text{Alice}}}$.

Let $V = b_{\text{Alice}} \cdot b^{-1}_{\text{Bob}} \bmod p$ such that $C'_1 = C_1^V = (g^r)^{b_{\text{Alice}} \cdot b^{-1}_{\text{Bob}}} \bmod p$. Thus when Bob derives $K' = (C'_1)^{b_{\text{Bob}}}$, we have

$$
\begin{aligned}
&(C'_1)^{b_{\text{Bob}}} \bmod p \\
&= ((g^r)^{b_{\text{Alice}} \cdot b^{-1}_{\text{Bob}}})^{b_{\text{Bob}}} \bmod p \\
&= (g^r)^{b_{\text{Alice}}} \bmod p
\end{aligned}
$$

as required (note the exponent cancellation works out due to the requirement $\gcd(b, p-1) = 1$ in the protocol setup).

Q7.6 (3 points) Recall that the ElGamal scheme from lecture defines $C_2 = M \cdot B^r \bmod p$ instead of $\mathsf{Enc}(H(B^r \bmod p), M)$.

Is it still possible to create a proxy re-encryption scheme with lecture ElGamal?

- ● (A) Yes, with an identical setup
- ○ (B) Yes, but with a modified setup
- ○ (C) No

Solution: The proxy re-encryption only deals with keeping the value of K the same, so changing how K is used to encrypt M is technically irrelevant. We can use the same setup as before without any changes.

Post-Exam Activity

EvanBot is having a post-midterm party! What did they cook?

Artwork by Shigezaki *Interested in having your art featured? Email evanbot@berkeley.edu.*

Comments/Assumptions Box

Congratulations for making it to the end of the exam! Feel free to leave any thoughts, comments, feedback, or doodles here. These comments won't affect your grade.

If there's anything else you want us to know, or you feel like there was an ambiguity in the exam, please put it in the box below. For ambiguities, you must qualify your answer and provide an answer for both interpretations. For example, "if the question is asking about A, then my answer is X, but if the question is asking about B, then my answer is Y". You will only receive credit if it is a genuine ambiguity and both of your answers are correct. We will only look at ambiguities if you request a regrade.