Weaver
Fall 2020

CS 161
Computer Security

Midterm

For questions with **circular bubbles**, you may select exactly *one* choice on Gradescope.

○ Unselected option

● Only one selected option

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

■ You can select

■ multiple squares

For questions with a **large box**, you need write and label your answer in the corresponding text box on Gradescope.

You have 110 minutes, plus a 10 minute buffer, for a total of 120 minutes. There are 7 questions of varying credit (120 points total).

The exam is open note. You can use an unlimited number of handwritten cheat sheets, but you must work alone.

Clarifications will be posted at https://cs161.org/clarifications.

Q1 *MANDATORY – Honor Code* (2 points)

Read the following honor code and type your name on Gradescope. ***Failure to do so will result in a grade of 0 for this exam.***

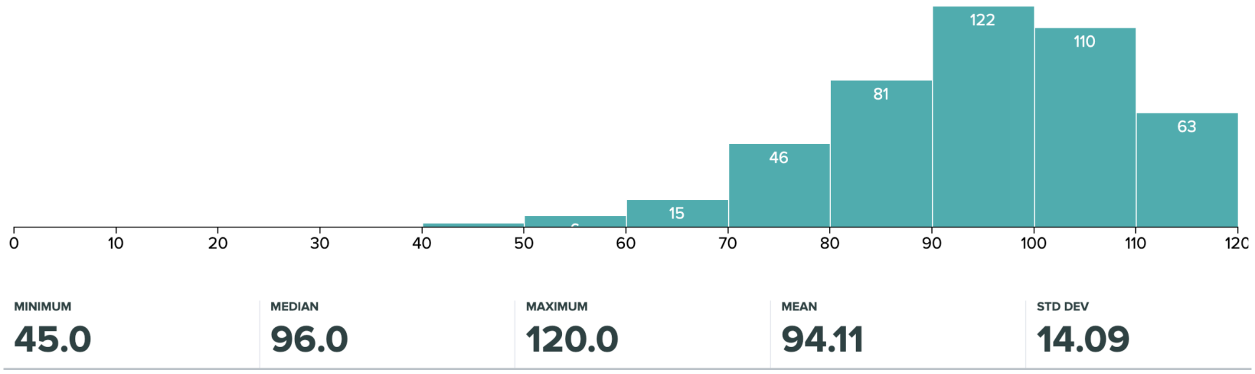
I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the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

Solution: Don't worry if you forgot to fill in your name. Everyone gets 2 free points for embracing the suck this semester.

We also won't take any points off if you entered something in a text box for a multiple-choice question, or if you bubbled in some options for a free-response question, or if you filled something in for a question that doesn't exist on your randomized form. To be consistent, we will not consider any unnecessary writing/bubbling on your exam during grading (pretend it's scratch work).

This is the end of Q1. Proceed to Q2 on your answer sheet.

Grade distribution (out of 120 points):



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
45.0	96.0	120.0	94.11	14.09

Q2 *True/false* (30 points)

Each true/false is worth 2 points.

Q2.1 TRUE or FALSE: Pointer authentication prevents all buffer overflow attacks.

○ TRUE ● FALSE

Solution: False. If you never overwrite a pointer (e.g. rip, sfp, program-allocated pointer on the stack), your exploit won't be detected. Remember, the only way to prevent all buffer overflow attacks is to use a memory-safe language.

Q2.2 TRUE or FALSE: The RET2ESP (Return to ESP) exploit from Project 1, Question 6 does **not** require knowing the **absolute** address of the shellcode when crafting the exploit.

● TRUE ○ FALSE

Solution: True. The RET2ESP technique works even when ASLR is enabled, a case in which we don't know what the exact address of the shellcode is.

Q2.3 TRUE or FALSE: The function $f(x) = 1$ is a one-way function, since we can't go from 1 to our original value of x.

○ TRUE ● FALSE

Solution: False. One-way functions are defined as "given $f(x) = y$, it is hard to find any x' such that $f(x') = y'$. This is false for $f(x) = 1$, since you can pick any value and satisfy $f(x) = f(x')$.

Q2.4 TRUE or FALSE: EvanBot designs custom buffer overflow protection that blocks all writes to RIP's and SFP's. This is a successful defense against all buffer overflow attacks.

○ TRUE ● FALSE

Solution: False. We can still overwrite other values on our stack (e.g. an "authenticated" flag).

Q2.5 TRUE or FALSE: Using `fgets(buf, size, ...)` instead of `gets(buf)` always prevents an attacker from overflowing `buf`.

○ TRUE ● FALSE

Solution: False. It's still possible for us to have an integer conversion vulnerability, where the value of `size` is greater than the actual value of the buffer (we saw this in the project!).

Many people asked in clarifications whether `size` is guaranteed to match the size of `buf`, which caused Nick to say..."Do *you* always use `fgets` correctly?"

Q2.6 TRUE or FALSE: Diffie-Hellman is a protocol for sending messages confidentially between two people who don't share a key.

○ TRUE ● FALSE

Solution: False. Diffie-Hellman is a key exchange protocol that allows two people to agree on a shared secret key. There is no message being sent in the Diffie-Hellman protocol.

Q2.7 TRUE or FALSE: The El Gamal protocol from lecture guarantees integrity.

○ TRUE ● FALSE

Solution: False. As seen in Homework 2, an attacker could change the ciphertext (c_1, c_2) to be $(c_1, 2c_2)$, causing the recipient to see the message $2x$ instead of x. There is no way for the recipient to detect this, so El Gamal does not guarantee integrity.

Q2.8 TRUE or FALSE: When using CBC mode, we need to pad messages because the block cipher takes a fixed-length input.

● TRUE ○ FALSE

Solution: True. As seen in Homework 2 and Lab 1, CBC mode breaks the plaintext up into blocks and passes each block through block cipher encryption, and since the block cipher takes a fixed-length input, we need the plaintext length to be a multiple of the block size.

Q2.9 TRUE or FALSE: Kerckhoffs's principle assumes that everything about a cryptographic system, including the key, is public knowledge.

○ TRUE ● FALSE

Solution: False. Everything *except* the key is public knowledge.

Q2.10 TRUE or FALSE: Slower hashes are useful for password hashing.

● TRUE ○ FALSE

Solution: True. A slower hash only costs a legitimate user an additional fraction of a second to check their legitimate password, but it increases the cost for an attacker performing a dictionary/brute-force attack by a huge constant factor.

Q2.11 TRUE or FALSE: In a digital signature scheme, the verifying key is private, and the signing key is public.

○ True ● False

Solution: False. The signing key is private, so only the owner of the signing key can generate valid signatures. The verifying key is public, so everyone can verify signatures.

Q2.12 True or False: A 64-bit stack canary on a 64-bit processor provides more protection than a 32-bit stack canary on a 32-bit processor.

● True ○ False

Solution: True. A 64-bit random canary is harder to guess by brute-force than a 32-bit random canary. There is no security disadvantage from having a longer stack canary (although there might be negligible performance impact).

Q2.13 True or False: Security is economics, so you should generally not use a $100 lock to secure a $10 product.

● True ○ False

Solution: True. As seen in Homework 1, a rational consumer should not spend more on the lock than the product.

Q2.14 True or False: The confidentiality of El Gamal is compromised if r, the random value chosen for each message sent, is public.

● True ○ False

Solution: True. The attacker can recover the original message by multiplying c_2 by A^{-r}.

Q2.15 True or False: RSA encryption without padding is IND-CPA secure.

○ True ● False

Solution: False. RSA encryption without padding, as seen in lecture (and CS70), is not CPA secure because it is deterministic.

This is the end of Q2. Proceed to Q3 on your answer sheet.

Q3 *MAC Madness* (18 points)

Evan wants to store a list of every CS161 student's firstname and lastname, but he is afraid Mallory will tamper with his list.

Evan is considering adding a cryptographic value to each record to ensure its integrity. For each scheme, determine what Mallory can do without being detected.

Assume MAC is a secure MAC, H is a cryptographic hash, and Mallory does not know Evan's secret key k. Assume that firstname and lastname are all lowercase and alphabetic (no numbers or special characters), and concatenation does not add any delimiter (e.g. a space or tab), so $\mathsf{nick} \| \mathsf{weaver} = \mathsf{nickweaver}$.

Clarifications during the exam: Bob is storing the names with the cryptographic value in the database. Duplicate records are not allowed. Mallory can change anything in the database. "A value of her choosing" means any arbitrary value.

Q3.1 (3 points) $\mathsf{H}(\mathsf{firstname} \| \mathsf{lastname})$

● (A) Mallory can modify a record to be a value of her choosing

○ (B) Mallory can modify a record to be a specific value (not necessarily of her choosing)

○ (C) Mallory cannot modify a record without being detected

○ (D) ——

○ (E) ——

○ (F) ——

Solution: Anybody can hash a value, so Mallory could change a record to be whatever she wants and compute the hash of her new record.

Q3.2 (3 points) $\mathsf{MAC}(k, \mathsf{firstname} \| \mathsf{lastname})$

Hint: Can you think of two different records that would have the same MAC?

○ (G) Mallory can modify a record to be a value of her choosing

● (H) Mallory can modify a record to be a specific value (not necessarily of her choosing)

○ (I) Mallory cannot modify a record without being detected

○ (J) ——

○ (K) ——

○ (L) ——

Solution: Because the concatenation doesn't have any indicator of where the first name ends and the last name begins, Mallory could shift some letters between the first name and last name. For example, she could change the name Nick Weaver to Ni Ckweaver, Nic Kweaver, Nickw Eaver, etc. Since the MAC would remain unchanged, this edit would be undetectable.

Q3.3 (3 points) $\mathsf{MAC}(k, \mathsf{firstname} \| \texttt{"-"} \| \mathsf{lastname})$, where "-" is a hyphen character.

○ (A) Mallory can modify a record to be a value of her choosing

○ (B) Mallory can modify a record to be a specific value (not necessarily of her choosing)

● (C) Mallory cannot modify a record without being detected

○ (D) —

○ (E) —

○ (F) —

Solution: Now, the concatenation includes a separator between first name and last name, so the attack from the previous part is no longer possible. Note that names are alphabetical, so they would never include a dash in them.

Q3.4 (3 points) $\mathsf{MAC}(k, \mathsf{H}(\mathsf{firstname}) \| \mathsf{H}(\mathsf{lastname}))$

○ (G) Mallory can modify a record to be a value of her choosing

○ (H) Mallory can modify a record to be a specific value (not necessarily of her choosing)

● (I) Mallory cannot modify a record without being detected

○ (J) —

○ (K) —

○ (L) —

Solution: Hashes have fixed-length output, so the attack from the previous part (shifting letters between the first and last name) is not possible here either. It will always be unambiguous where the first hash ends and the second hash begins.

Also, since both hashes are used as input to a single MAC, there is no way for an attacker without the key to generate a valid MAC for any different name.

Q3.5 (3 points) $\mathsf{MAC}(k, \mathsf{firstname}) \| \mathsf{MAC}(k, \mathsf{lastname})$

○ (A) Mallory can modify a record to be a value of her choosing

● (B) Mallory can modify a record to be a specific value (not necessarily of her choosing)

○ (C) Mallory cannot modify a record without being detected

○ (D) ——

○ (E) ——

○ (F) ——

Solution: Because the first name and last name have separate MACs, Mallory could swap the first name and last name, and swap the two halves of the MAC.

In other words, Mallory could change the name Nick Weaver to Weaver Nick, and change the MAC from $\mathsf{MAC}(k, \mathsf{nick}) \| \mathsf{MAC}(k, \mathsf{weaver})$ to $\mathsf{MAC}(k, \mathsf{weaver}) \| \mathsf{MAC}(k, \mathsf{nick})$.

Q3.6 (3 points) Which of Evan's schemes guarantee confidentiality on his records?

○ (G) All 5 schemes

○ (H) Only the schemes with a MAC

○ (I) Only the schemes with a hash

● (J) None of the schemes

○ (K) ——

○ (L) ——

Solution: MACs and hashes do not have any confidentiality guarantees.

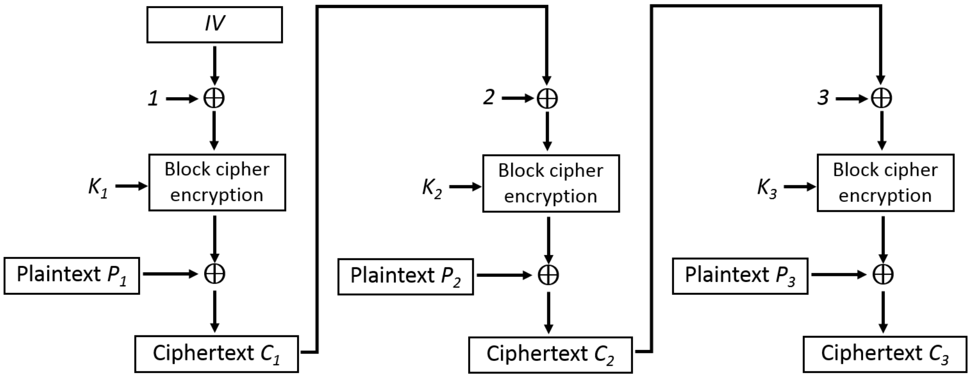
This is the end of Q3. Proceed to Q4 on your answer sheet.

Q4 *Socially Distanced Cipher* (18 points)

Bob and Alice want to plan a social distancing picnic, but don't want to invite Eve because she hasn't been wearing a mask in public. They decide to send messages using a new block cipher chaining mode, AES-SDC (Socially Distanced Cipher). Note that AES-SDC requires a different key for each block of the message.

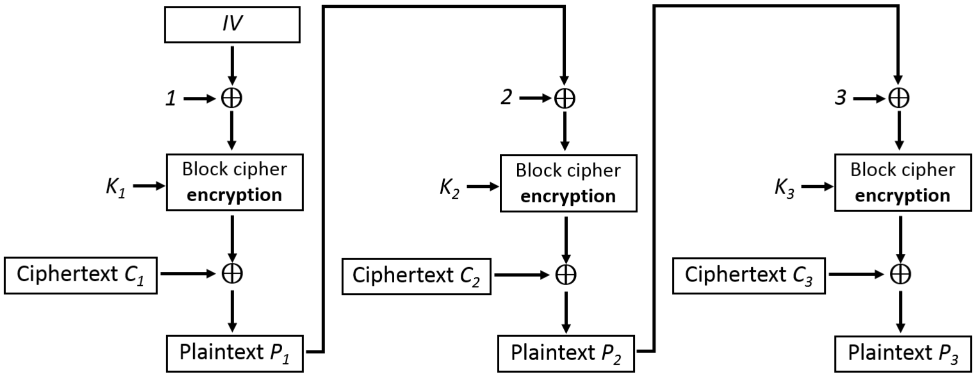
$$C_0 = IV$$

$$C_i = E_{k_i}(C_{i-1} \oplus i) \oplus P_i$$

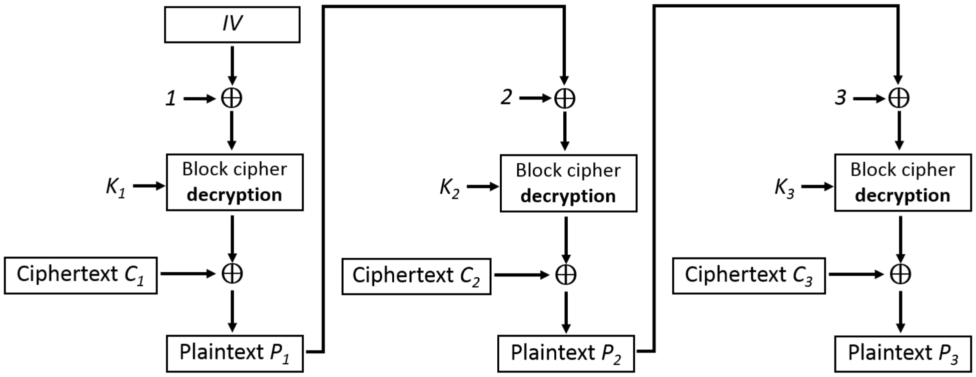


Q4.1 (3 points) Which of the following is the correct decryption expression/diagram for AES-SDC?

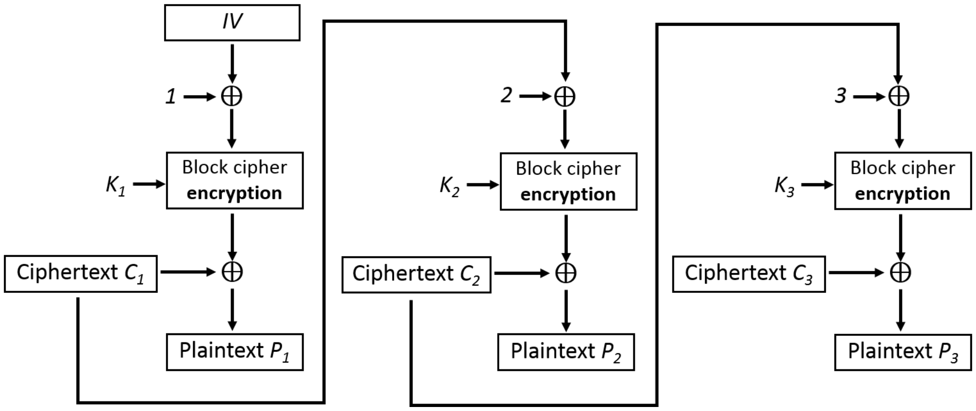
○ (A) $P_i = E_{k_i}(P_{i-1} \oplus i) \oplus C_i$



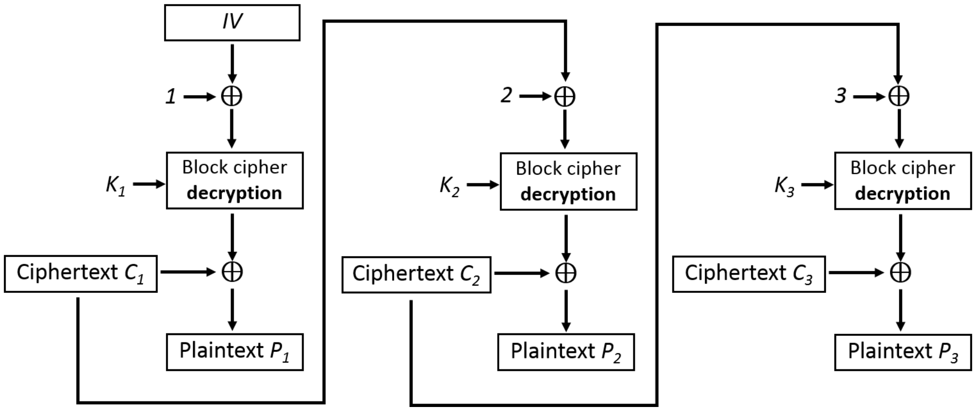
○ (B) $P_i = D_{k_i}(P_{i-1} \oplus i) \oplus C_i$



● (C) $P_i = E_{k_i}(C_{i-1} \oplus i) \oplus C_i$



○ (D) $P_i = D_{k_i}(C_{i-1} \oplus i) \oplus C_i$



○ (E) ——

○ (F) ——

Solution: In equations: To solve for P_i, XOR both sides of the encryption expression: $C_i \oplus E_{k_i}(C_{i-1} \oplus i) = P_i$.

In pictures: Observing the three-way XOR junctions, we see that to retrieve the plaintext, we need to XOR the ciphertext with the block cipher encryption (not decryption) output. This rules out options (B) and (D). The input to the block cipher encryption is the ciphertext, not the plaintext, which rules out options (A) and (B). Thus option (C) is the correct answer.

Q4.2 (3 points) Select all true statements about this encryption scheme.

Hint: The cipher mode you saw in Homework 2, $C_i = E_k(C_{i-1}) \oplus P_i$, is IND-CPA secure.

□ (G) Encryption can be parallelized

■ (I) It is IND-CPA secure

■ (H) Decryption can be parallelized

□ (J) None of the above

□ (K) ——　　　　□ (L) ——

Solution: Encryption cannot be parallelized, because calculating a ciphertext block C_i requires the previous ciphertext block C_{i-1} to be calculated first.

Decryption can be parallelized, because calculating a plaintext block P_i only requires ciphertext blocks C_i and C_{i-1}, which are already known before decryption starts.

The scheme is IND-CPA secure. Intuitively, AES-SDC is the same as the cipher mode from Homework 2, with two differences. First, a different key is used for each block cipher. This doesn't affect IND-CPA security because the attacker still doesn't know any of the secret keys. Second, a counter is added before encryption. This also doesn't affect IND-CPA security, since the output of a block cipher looks random to an attacker without the key, regardless of whether the input is XOR'd with a counter.

Suppose Alice loses some of her shared keys with Bob. Alice wants to encrypt an n-block message using AES-SDC. For each scenario below, determine which blocks Alice can still encrypt.

Q4.3 (3 points) Alice has all the keys except k_4 and k_5.

● (A) Alice can encrypt all parts of her message except P_4 and P_5

● (B) Alice can encrypt P_1, P_2 and P_3 only.

○ (C) Alice can encrypt the entire message

○ (D) Alice cannot encrypt any block of the message

○ (E) None of the above

○ (F) ——

Solution: The intended answer was (B). Without k_4 and k_5, Alice cannot run the block cipher encryptions needed to output C_4 and C_5. Since C_5 is used as an input to further block cipher encryptions, all blocks after C_5 cannot be encrypted either.

Someone in clarifications found an alternate solution to this part. If Alice doesn't know k_4 and k_5 and substitutes random/garbage values for the missing keys, then the corresponding ciphertexts C_4 and C_5 end up being random garbage as well. However, since C_5 is random garbage and is fed to the next encryption block, it can be used as an IV for future encryptions, which causes the rest of the encryption to be correct, even if Bob doesn't know what garbage values Alice used for k_4 and k_5!

We accepted both (A) and (B) as correct answers for full credit.

Now, suppose Alice now has all the keys, and Alice sends a n-block message to Bob. Eve learns some keys and some blocks of ciphertext. For each scenario below, determine which blocks Eve can decrypt.

Clarification during exam: Eve knows the index of all keys and ciphertext blocks that she learns.

Q4.4 (3 points) Eve learns the IV, ciphertext blocks C_5 and C_6, and key k_5.

○ (G) Eve can decrypt C_5 only

○ (H) Eve can decrypt C_5 and C_6 only

○ (I) Eve can decrypt all messages intercepted

● (J) Eve cannot decrypt any intercepted messages

○ (K) None of the above

○ (L) ——

Solution: In order to decrypt a ciphertext C_i, Eve needs to gain access to both C_{i-1} as well as k_i.

To decrypt C_5, Eve would need k_6, which she doesn't have, and to decrypt C_6, Eve would need k_7, which she also doesn't have. Thus Eve can't decrypt any intercepted messages.

Someone in clarifications noted that options H and I are technically equivalent, since C_5 and C_6 are the only messages Eve intercepts. However, neither is the correct answer choice, so this didn't affect our grading. Sorry if this caused any confusion.

Q4.5 (3 points) Eve learns the IV, ciphertext blocks C_2, C_3, and C_5, and keys k_2, k_3, and k_5.

○ (A) Eve can decrypt C_3 and C_5 only

○ (B) Eve can decrypt C_2, C_3, C_5 only

○ (C) Eve can decrypt C_2, C_3, C_4, C_5 only

● (D) Eve can decrypt C_3 only

○ (E) Eve cannot decrypt any intercepted messages

○ (F) None of the above

Solution: Using the same reasoning as the previous part, Eve has C_2, C_3, and k_3, so she can decrypt C_3.

To decrypt C_2, Eve would need C_1, which she doesn't have, and to decrypt C_5, Eve would need C_4, which she also doesn't have.

Q4.6 (3 points) Bob receives all the keys and ciphertext blocks C_1 through C_n, but C_3 is corrupted. Which plaintext blocks can Bob successfully decrypt?

Clarification during exam: "Bob receives all the keys and ciphertext blocks C_1 through C_n" should be "ciphertext blocks C_0 through C_n."

○ (G) Bob can successfully decrypt all blocks except C_3

○ (H) Bob can successfully decrypt all blocks except C_4

○ (I) Bob can successfully decrypt all blocks except C_1, C_2, C_3

● (J) Bob can successfully decrypt all blocks except C_3 and C_4

○ (K) Bob cannot successfully decrypt any of the blocks

○ (L) None of the above

Solution: The same reasoning from the previous parts applies here as well, where Bob has all the keys and all the ciphertexts, except C_3.

C_3 is needed in the decryption of C_3 and C_4, so Bob can't decrypt these two blocks. Bob can decrypt all other blocks.

This is the end of Q4. Proceed to Q5 on your answer sheet.

Q5 *Hacked EvanBot* (16 points)

Hacked EvanBot is running code to violate students' privacy, and it's up to you to disable it before it's too late!

```
1  #include <stdio.h>
2
3  void spy_on_students(void) {
4      char buffer[16];
5      fread(buffer, 1, 24, stdin);
6  }
7
8  int main() {
9      spy_on_students();
10     return 0;
11 }
```

The shutdown code for Hacked EvanBot is located at address `0xdeadbeef`, but there's just one problem—Bot has learned a new memory safety defense. Before returning from a function, it will check that its saved return address (rip) is not `0xdeadbeef`, and throw an error if the rip is `0xdeadbeef`.

Clarification during exam: Assume little-endian x86 for all questions.

Assume all x86 instructions are 8 bytes long. [1]Assume all compiler optimizations and buffer overflow defenses are disabled.

The address of `buffer` is `0xbffff110`.

Q5.1 (3 points) In the next 3 subparts, you'll supply a malicious input to the `fread` call at line 5 that causes the program to execute instructions at `0xdeadbeef`, *without* overwriting the rip with the value `0xdeadbeef`.

The first part of your input should be a single assembly instruction. What is the instruction? x86 pseudocode or a brief description of what the instruction should do (5 words max) is fine.

> **Solution:** `jmp *0xdeadbeef`
>
> You can't overwrite the rip with `0xdeadbeef`, but you can still overwrite the rip to point at arbitrary instructions located somewhere else. The idea here is to overwrite the rip to execute instructions in the buffer, and write a single jump instruction that starts executing code at `0xdeadbeef`.
>
> Grading: most likely all or nothing, with some leniency as long as you mention something about jumping to address `0xdeadbeef`. We will consider alternate solutions, though.

Q5.2 (3 points) The second part of your input should be some garbage bytes. How many garbage bytes do you need to write?

[1] In practice, x86 instructions are variable-length.

○ (G) 0 ○ (H) 4 ○ (I) 8 ● (J) 12 ○ (K) 16 ○ (L) ——

Solution: After the 8-byte instruction from the previous part, we need another 8 bytes to fill buffer, and then another 4 bytes to overwrite the sfp, for a total of 12 garbage bytes.

Q5.3 (3 points) What are the last 4 bytes of your input? Write your answer in Project 1 Python syntax, e.g. `\x12\x34\x56\x78`.

Solution: `\x10\xf1\xff\xbf`

This is the address of the jump instruction at the beginning of `buffer`. (The address may be slightly different on randomized versions of this exam.)

Partial credit for writing the address backwards.

Q5.4 (3 points) When does your exploit start executing instructions at `0xdeadbeef`?

○ (G) Immediately when the program starts

○ (H) When the `main` function returns

● (I) When the `spy_on_students` function returns

○ (J) When the `fread` function returns

○ (K) ——

○ (L) ——

Solution: The exploit overwrites the rip of `spy_on_students`, so when the `spy_on_students` function returns, the program will jump to the overwritten rip and start executing arbitrary instructions.

Q5.5 (4 points) Which of the following defenses would stop your exploit from the previous parts?

■ (A) Non-executable pages (also called DEP, WˆX, and the NX bit)

■ (B) Stack canaries

■ (C) ASLR

■ (D) Rewrite the code in a memory-safe language

□ (E) None of the above

□ (F) ——

Solution: Non-executable pages prevents the exploit because the exploit requires executing the `jmp` instruction that was written on the stack.

Stack canaries prevent the exploit because the exploit will overwrite the canary between `buffer` and the rip.

ASLR prevents the exploit because the exploit requires overwriting the rip with a known address on the stack.

Many people asked in clarifications if ASLR would change the address of the shutdown code at `0xdeadbeef`. We didn't answer this clarification because it doesn't affect the correct answer choice here: even if you knew the absolute address of the shutdown code, you couldn't overwrite the rip with the address of the buffer on the stack, because ASLR would randomize addresses on the stack.

Using a memory-safe language always prevents buffer overflow attacks.

This is the end of Q5. Proceed to Q6 on your answer sheet.

Q6 *Chegg* (17 points)

Engineers at Chegg are analyzing different password management techniques. Unfortunately, the engineers at Chegg used Chegg to make it through CS 161, so they don't remember anything they learned!

Q6.1 (3 points) Suppose there is an offline attacker (with access to the hashed passwords file) and an online attacker (without access to the hashed passwords file). Chegg implements a CAPTCHA on its login page. Which attacker(s) does the CAPTCHA prevent from performing a dictionary attack?

Clarification during exam: "prevent from performing a dictionary attack" means make the attack significantly more expensive.

○ (A) The offline attacker only

● (B) The online attacker only

○ (C) Both attackers

○ (D) Neither attacker

○ (E) ——

○ (F) ——

Solution: The CAPTCHA makes the attack significantly more difficult for the online attacker, who would have to fill out a CAPTCHA every time they try a guess at the password.

However, it would not stop the offline attacker, since they already have access to all the hashed passwords.

Q6.2 (3 points) Instead of salting each password hash, Chegg engineers XOR the hashed password with the account creation timestamp and store the XOR'd password hash with the timestamp in their database.

True or false: This successfully prevents an offline attacker from performing a dictionary attack.

Clarification during exam: The timestamp and the XOR'd password hash are both stored in the database. The offline attacker has the entire database.

○ (G) True ● (H) False ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Solution: False. An attacker could simply XOR each hash with the account creation timestamp to retrieve the original unsalted hash, which would be susceptible to dictionary attacks.

Q6.3 (4 points) One of Chegg's competitors, Course Hero, has been compromised, and all of their user accounts and passwords have been leaked in plaintext. Select all defenses that Chegg could use to protect students who use the same password for Chegg and Course Hero.

□ (A) Use a slow hash function

□ (B) Include a salt in the password hash e.g. store a tuple of $(\mathsf{salt}, \mathsf{H}(\mathsf{salt} \| \mathsf{password}))$

■ (C) Require every login attempt to also provide a random code sent by a secure SMS to the registered user's phone (a secure second factor)

□ (D) During the account creation phase, require every password to end with -CHEGG

□ (E) None of the above

□ (F) ——

Solution: A slow hash function, salted hashes, and passwords ending in -CHEGG all do nothing to stop an attacker who knows the plaintext passwords.

Two-factor authentication is the only valid defense here. Now, an attacker can't use the leaked passwords to log into the user's Chegg account, because they don't have the random code sent to the user's phone.

Some people asked in clarifications whether the attacker knows which password corresponds to which user. This doesn't change the answer, because even if the attacker doesn't have a mapping of users to passwords, the list of passwords still significantly reduces the search space for an attacker to try logins to Chegg accounts.

Chegg uses a certificate chain in order to verify tutors. When tutors post responses, they attach a digital signature of their response along with their certificate. Students can verify the authenticity of a response by verifying the certificate and using the public key in the certificate to verify the signature.

The certificate chain is below. Assume that the Chegg Root Certificate Authority (CA) is hardcoded into students' browsers.

1. Identity: Director of Chegg Recruiting (Verified by Chegg Root CA)
2. Identity: Campus Chegg Recruiter (Verified by Director of Chegg Recruiting)
3. Identity: Authorized Tutor (Verified by Campus Chegg Recruiter)

Q6.4 (4 points) EvanBot is not a valid tutor, but wants to create a fake tutor response with a valid signature. Which of these attacks would allow Bot to accomplish this?

□ (G) Steal the public key of the Campus Chegg Recruiter

■ (H) Steal the private key of the Director of Chegg Recruiting

■ (I) Steal the private key of the Chegg Root CA

□ (J) Steal the certificate of an authorized tutor

□ (K) None of the above

□ (L) ——

Solution: Public keys and certificates are public and well-known, so an attacker who steals them can't do anything to forge a message.

Stealing the private key of any individual in the certificate chain above the Authorized Tutor level allows EvanBot to create a forged certificate with a valid signature.

For example, EvanBot can use the private key of the Director of Chegg Recruiting to sign a certificate for a fake Campus Chegg Recruiter made up by EvanBot. Then, EvanBot uses the private key of the made-up Campus Chegg Recruiter to sign a certificate of a fake tutor. Finally, EvanBot can use the private key of the fake tutor to sign a tutor response.

Q6.5 (3 points) EvanBot gains access to the private key of Dave, who is an authorized tutor. Which of the following can EvanBot do?

□ (A) Post a valid response as Nick, an existing tutor

■ (B) Post a valid response as Dave

□ (C) Create and sign a certificate for Raluca, a new tutor

□ (D) None of the above

□ (E) ——

□ (F) ——

Solution: Dave's private key cannot be used to sign messages as Nick, who would have a different private key.

Dave's private key can be used to sign messages as Dave.

Dave's private key is at the tutor level (the lowest level of the certificate chain), so it cannot be used to sign certificates.

This is the end of Q6. Proceed to Q7 on your answer sheet.

Q7 *Stack Exchange* (19 points)

Consider the following vulnerable C code:

```
#include <byteswap.h>
#include <inttypes.h>
#include <stdio.h>

void prepare_input(void) {
    char buffer[64];
    int64_t *ptr;

    printf("What is the buffer?\n");
    fread(buffer, 1, 68, stdin);

    printf("What is the pointer?\n");
    fread(&ptr, 1, sizeof(uint64_t *), stdin);

    if (ptr < buffer || ptr >= buffer + 68) {
        printf("Pointer is outside buffer!");
        return;
    }

    /* Reverse 8 bytes of memory at the address ptr */
    *ptr = bswap_64(*ptr);
}

int main(void) {
    prepare_input();
    return 0;
}
```

The `bswap_64` function [2]takes in 8 bytes and returns the 8 bytes in reverse order.

Assume that the code is run on a 32-bit system, no memory safety defenses are enabled, and there are no exception handlers, saved registers, or compiler padding.

Q7.1 (3 points) Fill in the numbered blanks on the following stack diagram for `prepare_input`.

1	(0xbffff494)
2	(0xbffff490)
3	(0xbffff450)
4	(0xbffff44c)

○ (A) 1 = `sfp`, 2 = `rip`, 3 = `buffer`, 4 = `ptr`

○ (B) 1 = `sfp`, 2 = `rip`, 3 = `ptr`, 4 = `buffer`

● (C) 1 = `rip`, 2 = `sfp`, 3 = `buffer`, 4 = `ptr`

○ (D) 1 = `rip`, 2 = `sfp`, 3 = `ptr`, 4 = `buffer`

○ (E) ——

○ (F) ——

[2] Technically, this is a macro, not a function.

Solution: The rip is pushed onto the stack first, followed by the sfp, followed by the first local variable buffer, followed by the second local variable ptr. (Remember that local variables are placed on the stack, highest-to-lowest address, in the order they are defined in the code.)

Q7.2 (4 points) Which of these values on the stack can the attacker write to at lines 10 and 13? Select all that apply.

■ (G) `buffer`

■ (H) `ptr`

■ (I) `sfp`

□ (J) `rip`

□ (K) None of the above

□ (L) ——

Solution: At line 10, the attacker can write 68 bytes starting at `buffer`. This overwrites all 64 bytes `buffer` and the 4 bytes directly above it, which is the `sfp`.

At line 13, the attacker can write exactly 1 `uint64_t *` into `ptr`. This overwrites `ptr`, and nothing else.

Notice that the rip cannot be directly overwritten.

Q7.3 (3 points) Give an input that would cause this program to execute shellcode. At line 10, first input these bytes:

● (A) 64-byte shellcode

○ (B) `\xbf\xff\xf4\x4c`

○ (C) `\x4c\xf4\xff\xbf`

○ (D) `\xbf\xff\xf4\x50`

○ (E) `\x50\xf4\xff\xbf`

○ (F) ——

Q7.4 (3 points) Then input these bytes:

○ (G) 64-byte shellcode

○ (H) `\xbf\xff\xf4\x4c`

○ (I) `\x4c\xf4\xff\xbf`

● (J) `\xbf\xff\xf4\x50`

○ (K) `\x50\xf4\xff\xbf`

○ (L) ——

Q7.5 (3 points) At line 13, input these bytes:

○ (A) `\xbf\xff\xf4\x50`

○ (B) `\x50\xf4\xff\xbf`

○ (C) `\xbf\xff\xf4\x90`

● (D) `\x90\xf4\xff\xbf`

○ (E) `\xbf\xff\xf4\x94`

○ (F) `\x94\xf4\xff\xbf`

Solution: Line 10 writes 68 bytes into the 64-byte buffer, which lets us overwrite the sfp, but not the rip.

Line 13 lets us write a value into `ptr`, which is then dereferenced in a call to `bswap_64`. This lets us reverse any 8 bytes in memory we want, as long as they are between `buffer` and `buffer + 68`, i.e. in the buffer or sfp.

The overarching idea here is to write the address of shellcode in the sfp, and then use the call to `bswap_64` to swap the sfp and the rip.

First, we write the 64 bytes of shellcode into the buffer. Then, we overwrite the sfp with `\xbf\xff\xf4\x50`. These bytes are written backwards because `bswap_64` will reverse all 8 bytes of the sfp and the rip. Finally, we write the address of the sfp, `\x90\xf4\xff\xbf`, into ptr. These bytes are written normally because `bswap_64` never affects ptr.

Suppose the current rip is `0xdeadbeef`. Our input causes the 8 bytes starting at the sfp to be `\xbf\xff\xf4\x50\xef\xbe\xad\xde`. When we call `bswap_64` at the location of sfp, the 8 bytes starting at sfp are reversed, so they are now `\xde\xad\xbe\xef\x50\xf4\xff\xbf`. Notice that the rip is now pointing to the address of shellcode in the correct little-endian order. Also note that the original rip has been swapped into the sfp and is now backwards, although we don't care because the rip has already been overwritten.

Note: Because you can overwrite the sfp, you might be tempted to use the off-by-one exploit from Q4 of Project 1. However, this does not work here because you need enough space to write the shellcode and the address of shellcode in the buffer, but the buffer only has space for the shellcode.

Partial credit for Q7.4 option (K) and Q7.5 option (C) (correct address, but backwards).

Q7.6 (3 points) Suppose you replace `68` with `64` at line 10 and line 15. Is this modified code memory-safe?

○ (G) Yes ● (H) No ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Solution: This is still not memory-safe. If you make `ptr` point at one of the last 4 bytes of `buffer`, the check at line 15 will pass, but it will cause part of the sfp to be overwritten. For example, if `ptr` is located 4 bytes before the end of `buffer`, the last 4 bytes of `buffer` will be swapped into the sfp.

Because you can overwrite the sfp, you could still exploit this modified code using the technique from Project 1, Question 4 (although as mentioned above, you would need shorter shellcode). Regardless of what shellcode you use, since this code lets you write to the sfp (outside the bounds of `buffer`), it is not memory-safe.

This is the end of Q7. You have reached the end of the exam.

C Function Definitions

```
bswap_64(x);

       Returns a value in which the order of the bytes in its 8-byte argument
       is reversed.


char *fgets(char *s, int size, FILE *stream);

      fgets() reads in at most one less than size characters from stream and
      stores them into the buffer pointed to by s. Reading stops after an
      EOF or a newline.  If a newline is read, it is stored into the buffer.
      A terminating null byte ('\0') is stored after the last character in
      the buffer.


size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

       The function fread() reads nmemb items of data, each size bytes long,
       from the stream pointed to by stream, storing them at the location
       given by ptr.

       Note that fread() does not add a null byte after input.
```