Peyrin Kao
Fall 2022

CS 161
Computer Security

Midterm

Last updated: October 10, 2022 by FS

PRINT your name: _____, _____
(last) (first)

PRINT your student ID: _____

You have 110 minutes. There are 9 questions of varying credit (150 points total).

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	3	24	15	28	27	14	7	13	19	150

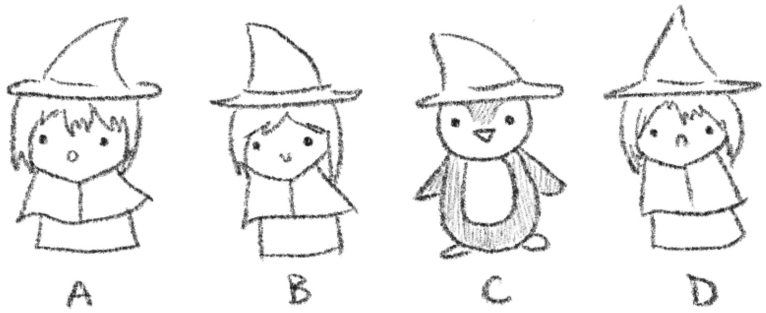
For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)

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

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

Pre-exam activity (not graded, just for fun): Circle the imposter. (Difficulty: Extra Hard.)



Q1 *Honor Code* (3 points)

Read the following honor code and sign your name.

> 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.

SIGN your name: _____

Q2 *True/False* **(24 points)**

Each true/false is worth 2 points.

Q2.1 True or False: If an attacker controls `buf`, then the line `printf("%s", buf)` contains a format string vulnerability.

○ True ● False

Solution: False. Format string vulnerabilities occur when the attacker controls the first input to `printf`.

Q2.2 True or False: If we notice that the stack canary has been changed, but the SFP and RIP above the stack canary are unchanged, then it is safe to continue executing the program.

○ True ● False

Solution: False. The stack canary being modified is a sign that your program is vulnerable, and someone may be trying to exploit that vulnerability. The safest thing to do here (and what C programs do in real life) is to crash the program immediately.

Q2.3 True or False: Non-executable pages, stack canaries, and ASLR are expensive defenses that should only be enabled if your program contains sensitive information.

○ True ● False

Solution: False. These defenses are effectively free on modern systems; there's no reason not to enable them.

Q2.4 True or False: If non-executable pages are enabled, it is impossible to execute shellcode on the heap.

● True ○ False

Solution: True. The heap is writable, so it's not executable.

Q2.5 Consider a variation of the IND-CPA game. Eve sends three plaintext messages to Alice. Alice randomly selects a message, encrypts it, and sends the encryption back to Eve.

True or False: If Eve cannot guess which message was encrypted with probability greater than $1/3$ (plus some negligible amount), then the scheme is IND-CPA secure.

● True ○ False

Solution: True. For Eve to break IND-CPA security, she must demonstrate a strategy that can correctly guess which message was encrypted with a better probability than randomly guessing. Eve can randomly guess which message was encrypted with probability $1/3$.

Q2.6 True or False: If an encryption scheme is not deterministic, then it must be IND-CPA secure.

○ True ● False

Solution: False. A non-deterministic scheme can still leak information about the plaintext. As a silly example, consider $\mathsf{Enc}(k, M) = M \| \text{some random value}$. This encryption scheme leaks everything about the message, but is non-deterministic.

Q2.7 True or False: A hash function whose output always ends in 0 cannot be a collision-resistant hash function.

○ True ● False

Solution: False. A simple example is to take a collision-resistant hash function and concatenate the end with 0.

Q2.8 Consider a PRNG that takes in a seed and generates pseudorandom output by outputting SHA-2(seed$\|x$), where x is a string of 0's that increases in length each time that output is generated (i.e. $x = 0$, then $x = 00$, then $x = 000$, etc.).

True or False: This is a secure PRNG.

○ True ● False

Solution: False. A length-extension attack can be used to predict future outputs of the PRNG.

Recall that in a length extension attack, an attacker who sees the output SHA-2(M) can construct SHA-2($M\|M'$) for any M' the attacker chooses, without ever knowing M. In this PRNG, given the first generated output SHA-2(seed$\|0$), an attacker could produce the second generated output SHA-2(seed$\|00$) with a length extension attack, without ever knowing seed.

Q2.9 True or False: In the Bitcoin protocol, digital signatures prevent an attacker from spending the same coin twice.

○ True ● False

Solution: False. Digital signatures prevent an attacker from spending other people's coins. Consensus via proof-of-work prevents an attacker from spending the same coin twice.

Q2.10 True or False: After a certificate authority places a certificate on a certificate revocation list, all clients immediately stop accepting the certificate as valid.

○ True ● False

Solution: False. Users might not download the revocation list until later.

Q2.11 True or False: The shorter the expiration date on certificates, the longer the certification revocation list.

○ True ● False

Solution: False. CRLs do not include certificates that are expired. In fact, a shorter expiration date on certificates tends to produce shorter CRLs, since there are fewer compromised, unexpired certificates on the CRL.

Q2.12 True or False: Since PRNGs are deterministic, their outputs are distinguishable from random to someone knows the internal state.

● True ○ False

Solution: True. Outputs are indistinguishable from random to someone who does not know the internal state of the PRNG. If an attacker knows the internal state of the PRNG, they could distinguish future PRNG output from random bits by computing future outputs of the PRNG.

Q3 *Trilogy* (15 points)

Alice, Bob, and CodaBot each generate a random secret: a, b, and c, respectively. They then compute $g^a \bmod p$, $g^b \bmod p$, and $g^c \bmod p$, respectively, and publish these values to everyone else.

CodaBot also generates an RSA key pair SK_C and PK_C. Assume that everyone else knows CodaBot's correct public key.

CodaBot sends a message M to both Alice and Bob. CodaBot also wants to send some additional value(s) to Alice and Bob to ensure authenticity on the message. Eve is a passive eavesdropper who sees the messages CodaBot is sending.

For each scheme below, select who is able to verify that the message came from CodaBot.

Q3.1 (3 points) CodaBot sends M and $\mathsf{HMAC}(c, M)$.

○ Alice only ○ Alice and Bob only ● Nobody

○ Bob only ○ Alice, Bob, and Eve

Solution: In order to verify the HMAC, the recipient would need to know the value of c, but only CodaBot knows this value.

Q3.2 (3 points) CodaBot sends M and $\mathsf{HMAC}(g^{ac} \bmod p, M)$ and $\mathsf{HMAC}(g^{bc} \bmod p, M)$.

○ Alice only ● Alice and Bob only ○ Nobody

○ Bob only ○ Alice, Bob, and Eve

Solution: Alice and Bob can use Diffie-Hellman to derive g^{ac} and g^{bc} respectively, then use those shared secrets to verify the HMAC. Note that Eve cannot derive these shared secrets, so she cannot verify the HMACs.

Q3.3 (3 points) CodaBot sends M and $\mathsf{HMAC}(g^{c(a+b)}, M)$.

○ Alice only ○ Alice and Bob only ● Nobody

○ Bob only ○ Alice, Bob, and Eve

Solution: Alice and Bob are unable to derive $g^{c(a+b)}$.

Q3.4 (3 points) CodaBot sends $C = \mathsf{Enc}(g^{ac} \bmod p, M)$ and $\mathsf{HMAC}(g^{ac} \bmod p, C)$.

- ● Alice only
- ○ Alice and Bob only
- ○ Nobody
- ○ Bob only
- ○ Alice, Bob, and Eve

Solution: g^{ac} is needed to verify the HMAC, and only Alice can compute this.

Q3.5 (3 points) CodaBot sends $C = \mathsf{Enc}(g^{ac} \bmod p, M)$ and $\mathsf{Sign}(SK_C, C)$.

Assume Enc refers to an IND-CPA secure encryption function.

- ○ Alice only
- ○ Alice and Bob only
- ○ Nobody
- ○ Bob only
- ● Alice, Bob, and Eve

Solution: Because the ciphertext is signed and everybody sees the ciphertext, everybody can verify the ciphertext's authenticity.

Q4 *So, you want a secure key?* (28 points)

Alice wants to create a secure channel of communication with a server. From CS 161, Alice remembers that the best way of communicating is to somehow end up with a shared, symmetric key, but has no idea how this process works.

Assume that there exists a certificate authority (CA) actively sending certificates to many clients. Assume that Mallory, a MITM attacker, and Eve, an eavesdropper, can both exist in all communication channels for all subparts unless otherwise specified.

Q4.1 (2 points) Alice first wants to authenticate the CA before authenticating the server. She remembers that certificates provide authenticity, so she exchanges the following messages with the CA:

1. Alice queries the CA for the CA's public key and receives PK_{CA}.
2. Alice queries the CA for the server's public key and receives $\{\text{“The server's public key is } PK_S\text{”}\}_{SK_{\text{CA}}^{-1}}$.

Can Mallory trick Alice into accepting a different public key PK'_S **of Mallory's choosing** as the server's public key without being detected?

● Yes ○ No

Solution: A MITM attacker could be present within the Alice–CA channel and therefore provide a public key (PK_{CA}) that is different from the CA's actual public key, to Alice. Since there is no root of trust, there is no way for Alice to know that she received an incorrect key.

For the rest of this question, assume that instead of querying the CA for their public key, Alice has the CA's correct public key, PK_{CA}, hardcoded into her computer.

Q4.2 (5 points) When Alice queries the CA for the server's public key, the CA sends $\{\text{“The server's public key is } PK_S\text{”}\}_{SK_{\text{CA}}^{-1}}$.

Can Mallory trick Alice into accepting a different public key PK'_S, **not necessarily of Mallory's choosing**, as the server's public key without being detected?

If you mark “Yes”, provide an attack that would accomplish this goal. If you mark “No”, explain why not in 2 sentences or fewer.

● Yes ○ No

Solution: Mallory can replay any certificate that has been sent over the channel previously. Two potential examples include a certificate of a different server that has previously been sent over the channel by the CA, or sending an old public key of *this* server.

Q4.3 (5 points) When Alice queries the CA for the server's public key, the CA selects a random number x between 1 and 20 and sends $\{\text{“The server's public key is } PK_S\text{”}\|x\}_{SK_{\text{CA}}^{-1}}$.

Can Mallory trick Alice into accepting a different public key PK'_S, **not necessarily of Mallory's choosing**, as the server's public key without being detected?

If you mark "Yes", provide an attack that would accomplish this goal. If you mark "No", explain why not in 2 sentences or fewer.

● Yes ○ No

Solution: The addition of x here doesn't really change anything from the previous subpart since the value of x can be repeated. An attacker could still replay a certificate that was sent over the channel.

Q4.4 (4 points) Alice has received some public key PK_S from the CA, but she doesn't trust that PK_S belongs to the server. Which of the following messages can the server send to convince Alice that she is talking to the legitimate server? Select all that apply.

☐ The server sends $H(PK_S)$

☐ The server sends $\mathsf{Sign}(SK_S, H(PK_S))$

☐ The server sends $H(SK_S||PK_S)$

☐ The server randomly generates a symmetric key K and sends $(\mathsf{Sign}(SK_S, K), \mathsf{HMAC}(K, PK_S))$

■ None of the above

Solution: This question is an issue of the server trying to authenticate themselves to Alice. If the server sends $H(PK_S)$, it doesn't authenticate themselves to Alice since any MITM could replace the hash with their own hash. If the server signs the hash of their public key, since Alice still does not have the server's actual public key, this doesn't work either. For example, an attacker who has sent Alice PK'_S could replace the signature with $\mathsf{Sign}(SK'_S, H(PK'_S))$. Finally, if the server generates a symmetric key, the same issue (and attack) with option 2 still applies.

For the rest of this question, assume that Alice knows the server's correct public key PK_S.

Q4.5 (4 points) Alice recalls that one of the best ways to come up with a shared, symmetric key is to use a key exchange protocol, so she decides to use Diffie-Hellman. If Alice and the server use the Diffie-Hellman protocol to derive a shared key K, which of the following statements must be true? Select all that apply.

- ☐ Alice and the server will always derive the same key, K.
- ☐ Eve can recover the private keys of Alice and the server.
- ■ The key exchange protocol provides forward secrecy against Eve.
- ■ If Eve records the protocol and then compromises Alice's secret Diffie-Hellman component a, she can derive K.
- ☐ None of the above

Solution: The question is asking about some properties of the Diffie-Hellman key exchange. The first answer choice cannot be correct since a MITM attacker could hijack the communications channel as seen in lecture. Since the discrete-log problem is hard, Eve, who only knows g^{as}, cannot factor it out and retrieve a or s. Diffie-Hellman provides forward secrecy, but if Eve compromises one of the Diffie-Hellman secret components, she can recalculate the shared symmetric key.

For the rest of this question, assume that Alice has a public-private key pair (SK_A, PK_A). Assume that Alice knows the server's correct public key PK_S, in addition to the server knowing Alice's correct public key PK_A.

Q4.6 (4 points) Alice wants to try a modified version of the Diffie-Hellman key exchange which is as follows:

- Alice sends $(g^a, \mathsf{Sign}(SK_A, g^a))$
- The server sends $(g^s, H(g^s))$
- The shared key is derived as $K = g^{as}$

Can Mallory trick the either Alice or the server into deriving a key that is different from the one they would have derived if Mallory had not existed?

If you mark "Yes", provide an attack that would accomplish this goal. If you mark "No", explain why not in 2 sentences or fewer.

● Yes ○ No

Solution: Alice signs her section of the exchange, but the server doesn't, meaning that a MITM attacker oculd replace whatever the server sends to be $(g^{s'}, H(g^{s'}))$ for an s' of the attacker's choosing.

Q4.7 (4 points) Alice gives up on independently deriving a shared key and instead decides to share keys instead. To share two randomly generated symmetric keys K_1 and K_2, Alice sends $C = (C_1, C_2)$, where

$$C_1 = \mathsf{PKEnc}(PK_S, K_1||K_2) \qquad C_2 = \mathsf{HMAC}(K_2, C_1)$$

Can Mallory trick either Alice or the server into deriving a shared key that is different from the one they would have derived if the Mallory had not existed?

If you mark "Yes", provide an attack that would accomplish this goal. If you mark "No", explain why not in 2 sentences or fewer.

● Yes ○ No

Solution: Since C_1 is encrypted using the server's public key, anyone can create an encryption message and send their own keys, $K_1' and K_2'$. They can then send an HMAC keyed with K_2' over their new ciphertext C_1'.

Q5 *AES-161* (27 points)

Alice has created a scheme called AES-161 to send messages to Bob securely in the presence of a man-in-the-middle attacker Mallory. Alice and Bob both share a symmetric key K that is secret from everyone else.

The encryption scheme for AES-161 is as follows:

$$C_1 = E_K(IV_1 \oplus M_1)$$
$$C_2 = E_K(C_1 \oplus IV_2 \oplus M_2)$$
$$C_i = E_K(C_{i-1} \oplus C_{i-2} \oplus M_i)$$

Q5.1 (3 points) Write the decryption formula of AES-161 for M_i, for $i > 2$.

Solution: $M_i = D_K(C_i) \oplus C_{i-1} \oplus C_{i-2}$

Solve this by beginning with the encryption formula for M_i and isolating M_i to its own side.

$$\begin{aligned}
&C_i = E_K(C_{i-1} \oplus C_{i-2} \oplus M_i) && \\
&D_K(C_i) = C_{i-1} \oplus C_{i-2} \oplus M_i && \text{(Call } D_K \text{ on both sides)} \\
&D_K(C_i) \oplus C_{i-1} = C_{i-2} \oplus M_i && \text{(XOR } C_{i-1} \text{ on both sides)} \\
&D_K(C_i) \oplus C_{i-1} \oplus C_{i-2} = M_i && \text{(XOR } C_{i-2} \text{ on both sides)}
\end{aligned}$$

Verify this solution:

$$\begin{aligned}
&D_K(C_i) \oplus C_{i-1} \oplus C_{i-2} \\
&= D_K(E_K(C_{i-1} \oplus C_{i-2} \oplus M_i)) \oplus C_{i-1} \oplus C_{i-2} \\
&= C_{i-1} \oplus C_{i-2} \oplus M_i \oplus C_{i-1} \oplus C_{i-2} \\
&= \cancel{C_{i-1}} \oplus \cancel{C_{i-2}} \oplus M_i \oplus \cancel{C_{i-1}} \oplus \cancel{C_{i-2}} \\
&= M_i
\end{aligned}$$

Q5.2 (4 points) Is this scheme IND-CPA secure with randomly generated IVs? If you mark "Yes", provide a brief justification (10 words or fewer; no formal proof necessary). If you mark "No", provide a strategy to win the IND-CPA game with probability greater than $1/2$.

● Yes ○ No

Solution: The scheme is IND-CPA secure with randomly generated IVs. Since E_K is a block cipher, its output is a pseudorandom permutation as long as the inputs are unique. Since the IVs are guaranteed to be random, even using the same messages will result in the input to the block cipher being unique, therefore implying that the output will be a pseudorandom permutation, making it indistinguishable from random.

Q5.3 (4 points) Select all true statements for messages longer than 2 blocks. Assume that the PRNG is a secure, rollback-resistant PRNG that has been seeded once with a constant, public value.

- ☐ AES-161 is IND-CPA secure if both IV_1 and IV_2 are generated as $H(i)$ where i is a global, monotonically increasing counter that is incremented after every encryption.
- ☐ AES-161 is IND-CPA secure if IV_1 is generated by generating bytes from the PRNG and IV_2 is generated as $\mathsf{HMAC}(K_2, IV_1)$.
- ■ AES-161 is IND-CPA secure if IV_1 is generated as $\mathsf{HMAC}(K_2, IV_2)$ and IV_2 is generated by generating bytes from the PRNG.
- ☐ AES-161 is IND-CPA secure if IV_1 is generated as $\mathsf{HMAC}(K_2, M_1)$ and IV_2 is $\mathsf{HMAC}(K_2, M_2)$.
- ☐ None of the above

Solution:
Option 1 is not a true statement. This is because both IV_1 and IV_2 are predictable to the attacker. Our scheme will lose IND-CPA security when IV_1 is predictable to an attacker. This is because an adversary playing the IND-CPA game can provide messages that cancel out IV_1 in the first block, making the first block deterministic.

Option 2 is not a true statement. In this scheme, IV_1 is generated using bytes from the PRNG that was seeded with a constant, public value. Since the PRNG was seeded with a constant, public value, the outputs of the PRNG are predictable, meaning that IV_1 is also predictable in this scheme. As mentioned in the previous statement, if IV_1 is predictable, then AES-161 is not IND-CPA.

Option 3 is a true statement. In this example, IV_2 is predictable (since the PRNG state is known to everyone because of the public seed), however, IV_1 is not predictable. In this scheme, the unpredictability of IV_1 alone is enough to keep IND-CPA security.

Option 4 is not a true statement. Generating IV_1 and IV_2 in this manner will make the scheme deterministic since the same message will always

Consider the following attack, called the FEI attack:

Given a ciphertext C of a known plaintext M, Mallory wishes to provide C' such that some subset of blocks of Mallory's choosing would be decrypted to M'_i, where both i and M'_i are **any values of Mallory's choosing**. For other values of i, the corresponding M'_is **can be anything**.

For example, let's say Mallory wants to provide a C' so that the first and last blocks of an 8-block message are decrypted into values M'_1 and M'_8 of her choosing while blocks 2 through 7 are not necessarily values of her choosing. In other words, when Bob decrypts the ciphertext C', he will get

$$M'_1 \| x_1 \| x_2 \| x_3 \| x_4 \| x_5 \| x_6 \| M'_8$$

where x_i refers to any value.

Q5.4 (6 points) Alice wishes to send a 3-block message M. Mallory wants to perform the FEI attack on the third block.

Provide a formula for all C'_i that differ from their corresponding C_i in terms of M_i, C_i, M'_i, and C'_i for specific values of i. Your formula may also include any public values. You don't need to provide a formula for any $C'_i = C_i$.

Solution: Any answer that sets $C'_3 = C_3$ and chooses C'_1 and C'_2 such that

$$C'_1 \oplus C'_2 = C_1 \oplus C_2 \oplus M_3 \oplus M'_3$$

is valid. We show this by running Bob's decryption process on the third block.

To get the plaintext for the third-block given C'_i, Bob will compute $D_K(C'_3) \oplus C'_2 \oplus C'_1$ (using the decryption formula from part one). Using our assumption that $C'_3 = C_3$ and $C'_1 \oplus C'_2 = C_1 \oplus C_2 \oplus M_3 \oplus M'_3$, we get that

$$\begin{aligned}
\text{Third Block's Plaintext} &= D_K(C'_3) \oplus C'_2 \oplus C'_1 \\
&= D_K(C_3) \oplus C_1 \oplus C_2 \oplus M_3 \oplus M'_3 \\
&= D_K(E_K(M_3 \oplus C_2 \oplus C_1)) \oplus C_1 \oplus C_2 \oplus M_3 \oplus M'_3 \\
&= M_3 \oplus C_2 \oplus C_1 \oplus C_1 \oplus C_2 \oplus M_3 \oplus M'_3 \\
&= \cancel{M_3} \oplus \cancel{C_2} \oplus \cancel{C_1} \oplus \cancel{C_1} \oplus \cancel{C_2} \oplus \cancel{M_3} \oplus M'_3 \\
&= M'_3
\end{aligned}$$

Thus, Bob will decrypt the third block as plaintext as M'_3 as desired.

Common Correct Answers:

- $C'_1 = C_1 \oplus M_3 \oplus M'_3$, $C'_2 = C_2$, $C'_3 = C_3$
- $C'_1 = C_1$, $C'_2 = C_2 \oplus M_3 \oplus M'_3$, $C'_3 = C_3$
- $C'_1 = C_1 \oplus C_2 \oplus M_3 \oplus M'_3$, $C'_2 = 0$, $C'_3 = C_3$
- $C'_1 = 0$, $C'_2 = C_1 \oplus C_2 \oplus M_3 \oplus M'_3$, $C'_3 = C_3$

Q5.5 (5 points) Assume that Alice is sending a 9-block message. What is the maximum number of blocks that Mallory can perform the FEI attack on?

Solution: 4. It is blocks 1, 3, 5, and 7. Or block 2, 4, 6, and 8.

Q5.6 (5 points) Assume that Alice is sending a 9-block message. Mallory wants to perform the FEI attack on the maximum number of blocks. You can pick which blocks the FEI attack is performed on.

Provide a formula for all C_i' that differ from their corresponding C_i in terms of M_i, C_i, M_i', and C_i' for specific values of i. Your formula may also include any public values. You don't need to provide a formula for any $C_i' = C_i$.

Solution: Our original intention for this question was to modify the IV to be $IV' = IV \oplus M_1 \oplus M_1'$ first. Then, the attack would be $C_i' = C_i \oplus M_{i+1} \oplus M_{i+1}'$ for i in [2, 4, 6, 8], thus making a total of 5 modified blocks.

However, we felt as though we did not make it clear enough that the IV could be modified, and therefore the actual solution for this problem would be to modify $C_i' = C_i \oplus M_{i+1} \oplus M_{i+1}'$ for one of the following:

- i in [2, 4, 6, 8]
- i in [1, 3, 5, 7]

Q6 *Group Chat* **(14 points)**

Recall that the ElGamal scheme from lecture is used to send a message to a single recipient using their public key. We would like to modify this scheme to work in a group chat, where one person can send a message, anyone in the group chat can decrypt the message, and no one outside of the group chat can decrypt the message.

Consider a four-person group chat consisting of Alice, Bob, Charlie, and David. Their private keys are a, b, c, and d. Their public keys are $A = g^a \bmod p$, $B = g^b \bmod p$, $C = g^c \bmod p$, and $D = g^d \bmod p$, respectively, known to everyone (including people outside the group chat).

Q6.1 (4 points) EvanBot proposes an encryption scheme: the four people in the group chat exchange messages to derive a shared value $g^{a+b+c+d+r} \bmod p$. Alice sends the tuple $(g^r, M \times g^{a+b+c+d+r})$ to the group chat.

Is this a valid scheme, where everybody in the group chat can decrypt the message, and no one outside the group chat can decrypt the message? Briefly justify your answer.

○ Yes ● No

Solution: Anybody can generate $g^{a+b+c+d}$ by multiplying the four public keys together and decrypt the message.

Q6.2 (2 points) Now consider a general group chat consisting of m users, where each message is n bits long. Each user i has a private key a_i, known only to themselves, and a public key $A_i = g^{a_i} \bmod p$, known by everyone.

Is it possible for Alice (who is user $i = 0$) to send a single message of length no more than $O(n)$ that is decryptable by everyone in the group chat but no one outside of the group chat?

○ Yes ● No

Solution: For Alice to send a message to everyone else of length no more than $O(n)$, she would have to have a single, shared secret known to everyone (which would be of length $O(n)$). The scheme described above would require length $O(mn)$, which is disallowed.

Q6.3 (4 points) Alice, Bob, Charlie, and David (with private keys a, b, c, and d) want to perform a shared key exchange to arrive at a shared key $g^{abcd} \bmod p$. What messages should they all send so that they all arrive at the same shared key, such that no eavesdropper can derive the value of the shared key?

Each message can only contain one value and one recipient, and each participant starts only knowing their private key. Use the format below. For example, if EvanBot is sending key g^e to Peyrin, it would be listed as follows:

Sender Evanbot Receiver Peyrin Message g^e

Not all rows may be used.

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Sender _____ Receiver _____ Message _____

Solution:

Sender Bob	Receiver Charlie	Message g^b
Sender Charlie	Receiver David	Message g^{bc}
Sender David	Receiver Alice	Message g^{bcd}
Sender Alice	Receiver Charlie	Message g^a
Sender Charlie	Receiver David	Message g^{ac}
Sender David	Receiver Bob	Message g^{acd}
Sender Alice	Receiver Bob	Message g^a
Sender Bob	Receiver David	Message g^{ab}
Sender David	Receiver Charlie	Message g^{abd}
Sender Bob	Receiver Charlie	Message g^{ab}
Sender Charlie	Receiver David	Message g^{abc}

Q6.4 (4 points) Alice, Bob, Charlie, and David have executed the key exchange from the previous subpart. Now, Alice realizes that she doesn't like David and doesn't want him to see her messages to Bob and Charlie.

Is it possible for Alice to send **one** message in the group chat such that Bob and Charlie can read it, but not David? Your message may not scale with the number of users in the group chat.

Assume that Alice, Bob, Charlie, and David can perform any number of key exchanges amongst themselves, but cannot generate any new keys.

○ Yes ● No

Briefly explain why or why not.

Solution: Once the key is created in the group-chat, the "sub-keys" are never going to be private within any subgroup. For example, g^{abc} will always be known to D, and D can use this value to decrypt any communications that exist between the rest of the group.

Q7 *Small Hulk* (7 points)

Consider the following vulnerable C code:

```
1 void hulk(char *eyes) {
2     char anger[16];
3     strcpy(anger, eyes);
4     printf("Hulk SMASH! %s\n", anger);
5 }
```

In this question, your goal is to delete the `smash.txt` file, which Hulk uses to smash his targets! Here are a few tools you can use:

- The `remove` standard C library method can be used to delete a file. The signature of the `remove` function is provided in the C appendix.
- The address of the `remove` function is `0xdeadbeef`.
- The address of `anger` is `0xffffdc10`.
- The string `"smash.txt"` exists in memory at `0xffffe644`.

Assume that **non-executable pages are enabled**, but all other memory safety defenses are disabled. Provide a string input to `eyes` that would delete `smash.txt`.

Solution: 'A' * 20 + '\xef\xde\xad\xde' + 'b' * 4 + '\x30\xdc\xff\xff' + 'smash.txt'

```
30 smash.txt
2c address of smash.txt
28 4 bytes of garbage
24 RIP hulk
20 SFP hulk
10 anger
```

Alternatively, placing 'smash.txt' within our buffer: 'smash.txt' + 'A' * 11 + '\xef\xde\xad\xde' + 'b' * 4 + '\x10\xdc\xff\xff'.

Finally, you could also use the address of 'smash.txt' that exists in memory to get: 'A' * 20 + '\xef\xde\xad\xde' + 'b' * 4 + '\44\e6\xff\xff'.

Q8 *Hulk Smash!* (13 points)

Assume that:

- For your inputs, you may use `SHELLCODE` as a 16-byte shellcode.
- If needed, you may use standard output as `OUTPUT`, slicing it using Python syntax.
- All x86 instructions are **4 bytes** long.
- For each provided code snippet, you run GDB once, and discover that:
  - The address of the RIP of the `hulk` method is `0xffffcd84`.
  - The address of a `ret` instruction is `0x080722d8`.

Consider the following function:

```
1  int hulk(FILE *f, char *eyes) {
2      void (* green_ptr)(void) = &green; //function pointer
3      char buf[32];
4      char str[28];
5      fread(buf, 1, 32, f);
6      printf("%s", buf);
7      fread(buf, 4, 32, stdin);
8      if (strlen(eyes) > 28) {
9          return 0;
10     }
11     strncpy(str, eyes, sizeof(buf));
12     return 1;
13 }
```

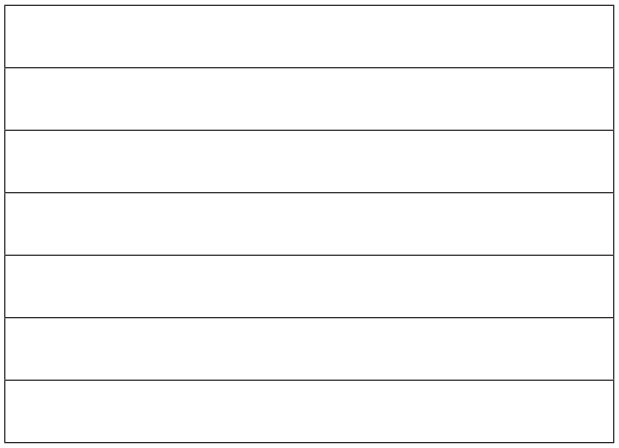
The following is the x86 code of `void green(void)`:

```
1 nop
2 nop
3 nop
4 ret
```

Assume that ASLR is enabled including the code section, but all other memory safety defenses are disabled.

Q8.1 (3 points) Fill in the following stack diagram, assuming that the program is paused after executing **Line 5**, including the arguments of `hulk` (the value in each row does not necessarily have to be four bytes long).

Stack



Solution: Stack diagram:

```
[4] char *eyes
[4] File* f
[4] RIP hulk
[4] SFP hulk
[4] (void)* green_ptr
[32] char buf
[28] char str
```

Q8.2 (10 points) Provide an input to each of the boxes below in order to execute `SHELLCODE`.

Provide a string value for `eyes` (argument to `hulk`):

Solution: SHELLCODE

Provide a string for the contents of the file that is passed in as the `f` argument of `hulk`:

Solution: 'A' * 32

Provide an input to the second `fread` in `hulk`:

Solution: 'A' * 40 + (OUTPUT[32:36] * 2)

Solution: Since ASLR is enabled, we need to leak an address on stack, and we notice that we have a function pointer with a ret instruction, so we can try to leak that address with the printf function. Recall that `printf` stops printing when it sees a null byte, so we have to get rid of all the null bytes between the beginning of the buffer and the function pointer. To slice the output, since we are printing from the start of `buf`, we can slice it from indices 32 to 36 as we notice that the green function is simply filled with NOPs and a ret instruction. In our second `fread`, we then start writing from the bottom of buf till we reach the RIP, and overwrite the RIP and (RIP+4) with ret instructions. We need two ret instructions here since we notice that we want some pointer that points to our SHELLCODE, which we placed in eyes.

Q9 *Brainf[REDACTED]* (19 points)

Consider the following code:

```
1  void execute(char* commands, FILE *file) {
2      int buf_ind = 0;
3      int buf_len = 16;
4      char buf[buf_len];
5      size_t comm_ind = 0;
6      while (commands[comm_ind]) {
7          if (commands[comm_ind] == 'C') {
8              buf_ind += 1;
9          } else if (commands[comm_ind] == 'D') {
10             buf_ind -= 1;
11         } else if (commands[comm_ind] == 'E') {
12             printf("%c", buf[buf_ind]);
13         } else if (commands[comm_ind] == 'F') {
14             printf("%x", &buf[buf_ind]);
15         } else if (commands[comm_ind] == 'G') {
16             fread(&buf[buf_ind], sizeof(char), 1, file);
17         }
18         /* assume you are provided two functions: min and max. */
19         buf_ind = max(0, min(buf_len, buf_ind));
20         comm_ind += 1;
21     }
22 }
```

For this question, assume the following:

- You may use `SHELLCODE` as a 52-byte shellcode.
- Stack canaries are enabled, and all other memory safety defenses are disabled.
- If needed, you may use the standard output as `OUTPUT`, slicing it using Python syntax.
- The RIP of `execute` is located at `0xffffabcc`.
- The top of the stack is located at `0xffffffff`.
- `execute` is called from `main` with the proper arguments.

Q9.1 (4 points) Fill in the following stack diagram, assuming that the program is paused after executing **Line 6**, including the arguments of `execute` (the value in each row does not necessarily have to be four bytes long).

Stack

Solution: Stack diagram:

```
[4] File *file
[4] char* commands
[4] RIP execute
[4] SFP execute
[4] canary
[4] buf_ind
[4] buf_len
[16] buf
[4] comm_ind
```

Q9.2 (12 points) We wish to construct a series of inputs that will cause this program to execute SHELLCODE that works 100% of the time.

Provide a string input to variable commands (argument to execute):

Solution: `'C' * 16 + 'G' + 'C' * 16 + 'GC' * 4 + 'C' * 8 + 'GC' * 52`

Provide a string for the contents of the file that is passed in as the file argument of execute:

Solution: `'\xff' + '\xd8\xab\xff\xff' + SHELLCODE`

Solution:
Exploit's Concept: The min/max logic on line 19 of the function introduces an off-by-one vulnerability, allowing buf_ind to go one index past the end of buf.

Using this off-by-one and the G command, we can change the LSB of buf_len to any value of our choosing. Thus, we can increase the value of buf_len from 16 to 255. Now buf_len can further up the stack.

This allows us to now move buf_ind up to the RIP of execute and overwrite it with the address of our shellcode, and then write SHELLCODE above the RIP of execute.

One thing to be careful about in the previous step is that SHELLCODE cannot be placed directly after the RIP of execute, because this will overwrite the commands and file pointers, causing the function to not process future commands and inputs correctly.

Exploit Construction: We begin with 16 C commands to move buf_ind to the LSB of buf_len. Next, we include one G command so that we read one byte from file and write it to the LSB of buf_len. We include an 0xff byte in file so that this G command will change buf_len from 0x00000010 (16) to 0x000000ff (255). Now buf_ind is allowed to move anywhere between 0 and 255 (inclusive).

Taking advantage of this, we can now use 16 C commands to move buf_ind to the RIP of execute. Next, we will use a G command to read a byte from file into the LSB of the RIP of execute. We want this byte from file to be the LSB of the address of our SHELLCODE, which we will calculate momentarily. We then use a C command to move buf_ind to the next byte of the RIP of execute. We repeat three more GC commands to write the three remaining bytes of RIP of execute.

Recall that we cannot directly write the SHELLCODE right after the RIP of execute because we cannot overwrite the arguments of execute. Thus, we must write SHELLCODE after the two arguments of execute, meaning buf_ind must go up 8 more bytes. Thus, we include another 8 C commands. Now, we can write SHELLCODE into memory using 52 repeated GC commands. The address SHELLCODE will be written at is above file, which is 12 bytes above the RIP of execute. We are given that the address of the RIP of execute is 0xffffabcc, so 12 bytes above this 0xffffabd8.

Q9.3 (3 points) If ASLR is now enabled, which of the following modifications to the provided code would allow you to execute `SHELLCODE` 100% of the time? Select all that apply.

- [x] Line 10 is replaced with `scanf("%u", &buf_ind)`.
- [ ] `jmp *esp` is located in your code at `0xdeadbeef`.
- [ ] Line 14 is replaced with `comm_ind = getchar()`.
- [ ] None of the above

Solution: We need two things to make ASLR work: we need to leak an address and we need to have a way to pause the program. We can pause the program with `scanf` and `getchar`. However, `getchar` gets rid of the line that leaks the address. `jmp *esp`, while it would normally work, overwrites the file pointer, which is required during the `fread` calls.

Nothing on this page will affect your grade in any way.

Activity: Zoo

EvanBot made a new friend at the zoo! What animal shall Bot befriend next?

Doodle

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