CS 161 Spring 2025 — Introduction to Computer Security — Final

Name: ______________________________

Student ID: ____________________________

This exam is 170 minutes long.

Question:	1	2	3	4	5	6
Points:	0	9	11	15	15	12

Question:	7	8	9	10		Total
Points:	10	10	7	11		100

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

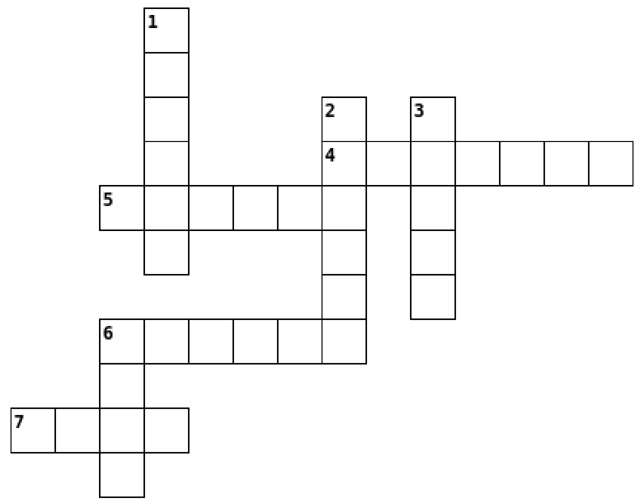
- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)
- Don't do this (it will be graded as incorrect)

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 - Crossword (0 points):



Across
4. Mascot who loves cookies
5. Parroting attack
6. ___ UNION, enemy of Caltopia
7. Default road sign password from lec. 1

Down
1. Someone who exploits systems
2. Our Lecturer
3. Shannon's ___
6. Insecure C input function

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* (9 points)

Each true/false is worth half of a point.

Q2.1 EvanBot decides to revamp their home network infrastructure with security in mind from the beginning of the design.

TRUE or FALSE: This is an example of using fail-safe defaults.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.2 TRUE or FALSE: Non-executable pages always mark the stack as executable.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.3 TRUE or FALSE: The off-by-one attack as seen in Project 1 involves overwriting the LSB of the RIP to point to the attacker's `SHELLCODE`.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.4 TRUE or FALSE: It is better to MAC-then-Encrypt rather than Encrypt-then-MAC, because the latter involves decrypting untrusted ciphertext before verifying integrity.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.5 TRUE or FALSE: In authenticated encryption, the same key should be used to both MAC and encrypt the message.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.6 TRUE or FALSE: For a block cipher mode of operation to be IND-CPA secure, it must be deterministic.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.7 TRUE or FALSE: Parameterized SQL is an effective defense against SQL injection.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.8 TRUE or FALSE: It is possible for a single cookie to be sent to two URLs with different origins.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.9 TRUE or FALSE: `https://google.com` and `https://google.com/maps` share the same origin.

- ○ (A) TRUE
- ○ (B) FALSE

Q2.10 TRUE or FALSE: In WPA2, an attacker who leaks only the value of PSK can find the WiFi password without using brute force.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.11 TRUE or FALSE: In TLS, a certificate is a signed message containing the server's domain, signed with the server's private key.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.12 TRUE or FALSE: After a TLS handshake completes, both parties use a single shared key to encrypt and MAC their messages.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.13 TRUE or FALSE: TLS can provide end-to-end encryption even when lower-level networking layers are compromised by a MITM.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.14 TRUE or FALSE: DNS uses UDP instead of TCP because UDP has increased speed and lower overhead compared to TCP.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.15 TRUE or FALSE: In DNS source port randomization, the name server's response packet has its source port field randomized to increase the difficulty of DNS spoofing.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.16 TRUE or FALSE: SYN cookies enable a server to store state only after the TCP handshake completes.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.17 TRUE or FALSE: Signature-based detection is effective at stopping new attacks.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.18 TRUE or FALSE: Polymorphic malware encrypts itself when propagating in order to obfuscate its source code.

- ◯ (A) TRUE
- ◯ (B) FALSE

Q2.19 (0 points) TRUE or FALSE: `EvanBot is a real bot.`

- ◯ (A) TRUE
- ◯ (B) FALSE

Q3 *Looping Into The Ocean* **(11 points)**

Consider the following vulnerable C code:

```c
1  void ocean(char* s, char* t) {
2      for (int i = 0; i < 20; i++) {
3          s[7-i] = t[i];
4      }
5  }
6
7  void whale() {
8      char tuna[20];
9      char salmon[8];
10
11     fread(tuna, 1, 20, stdin);
12     ocean(salmon, tuna);
13 }
14
15 int main() {
16     whale();
17     return 0;
18 }
```

RIP of `main`
SFP of `main`
RIP of `whale`
SFP of `whale`
`tuna`
(1)
`t`
(2)
(3)
SFP of `ocean`

Assumptions:

- All memory safety defenses are disabled.
- There is SHELLCODE stored at `0xDEADBEEF`.

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

- ◯ (A) (1) `tuna` (2) RIP of `ocean` (3) `s`
- ◯ (B) (1) `s` (2) `t` (3) RIP of `ocean`
- ◯ (C) (1) RIP of `ocean` (2) `s` (3) `t`
- ◯ (D) (1) `salmon` (2) `s` (3) RIP of `ocean`

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

- ◯ (A) Signed/unsigned
- ◯ (B) Out-of-bounds write
- ◯ (C) Time-of-check to time-of-use
- ◯ (D) Off-by-one

Q3.3 (3 points) Provide an input to the `fread` on Line 11 that will execute SHELLCODE.

- ◯ (A) `'A'*12 + '\xDE\xAD\xBE\xEF'`
- ◯ (B) `'A'*16 + '\xDE\xAD\xBE\xEF'`
- ◯ (C) `'\xDE\xAD\xBE\xEF' + 'A'*12`
- ◯ (D) `'A'*8 + '\xEF\xBE\xAD\xDE'`

Reminder: In a big-endian system, the most significant byte of a word is stored at the lowest memory address.

Consider a modified program running on a **big-endian** system, with the differences identified below:

```c
1  void ocean(char* s, char* t) {
2      for (int i = 0; i < 17; i++) { // modified
3          s[7-i] = t[i];
4      }
5  }
6
7  void whale() {
8      char tuna[20];
9      char salmon[8];
10
11     fread(tuna, 1, 17, stdin); // modified
12     ocean(salmon, tuna);
13 }
14
15 int main() {
16     whale();
17     return 0;
18 }
```

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

```
0x080010C4: push %ebp
0x080010C8: mov %esp, %ebp
...
0x08020010: pop %ebp
0x08020014: ret
```

Suppose that the RIP of `ocean` holds the value `0x080200C4`, and you want to execute SHELLCODE at `0xDEADBEEF`.

Q3.4 (1 point) What type of memory safety exploit is this code vulnerable to?

- ○ (A) ret2ret
- ○ (B) ret2libc
- ○ (C) Integer conversion
- ○ (D) `printf` vulnerability

Q3.5 (5 points) Give an input to the `fread` on Line 11 that executes SHELLCODE. If a part of the input can be any non-zero value, use `'A' * n` to represent `n` garbage bytes.

Q4 *printf("This looks familiar... ")* **(15 points)**

Consider the following vulnerable C code:

```c
void stack_editor(unsigned int num_commands) {
    char clipboard[4];
    char* arg_ptr = clipboard + 4;

    char* commands = malloc(num_commands + 1);
    fgets(commands, num_commands + 1, stdin);

    for (int i = 0; i < num_commands; i++) {
        char next_cmd = commands[i];

        if (next_cmd == 'C') { // Copy and Skip
            memcpy(clipboard, arg_ptr, 4);
            arg_ptr += 4;
        } else if (next_cmd == 'V') { // Paste and Skip
            memcpy(arg_ptr, clipboard, 4);
            arg_ptr += 4;
        } else if (next_cmd == 'D') { // Decrement
            (*((char*) arg_ptr)) -= 4;
        } else if (next_cmd == 'S') { // Skip 4 Bytes
            arg_ptr += 4;
        }
    }
    free(commands);
}

void main() {
    char sh_str[4] = "sh\0\0";

    system("ls -al");
    stack_editor(8);
}
```

HINT: The syntax `(*((char*) arg_ptr)) -= 4;` *goes to address* `arg_ptr` *in memory, and subtracts 4 from the value at that address.*

Assume ASLR and non-executable pages are enabled, but all other memory safety defenses are disabled.

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

```
0x08076030: call system
0x08076034: add $4, %esp
0x08076038: push $8
0x0807603C: call stack_editor
0x08076040: add $4, %esp
```

Q4.1 (1 point) Where does the SFP of `stack_editor` point to if the program is paused at Line 2?

- ◯ (A) SFP of `main`
- ◯ (B) commands
- ◯ (C) RIP of `stack_editor`
- ◯ (D) RIP of `stack_editor` + 4

Q4.2 (2 points) Suppose we run this program with input `DDCVSSSS` to the `fgets` on Line 6. Assume **for this subpart only** that the address of `clipboard` on the stack is `0xFFFFFF00`, and `0x08076000` is the value stored in RIP of `stack_editor`.

RIP of `main`		RIP of `main`
SFP of `main`		SFP of `main`
`sh_str`		`sh_str`
`num_commands`		`num_commands`
RIP of `stack_editor`		(1)
SFP of `stack_editor`	$\xrightarrow{\texttt{stack\_editor}}$	(2)
`clipboard`		`clipboard`
`arg_ptr`		`arg_ptr`
`commands`		`commands`
`next_cmd`		`next_cmd`

Fill in the values of the missing stack entries for the stack after the for-loop in `stack_editor` finishes executing, but before `stack_editor` returns.

- ◯ (A) (1) `0xFFFFFF0C` (2) `0xFFFFFF0C`
- ◯ (B) (1) `0xFFFFFF0C` (2) `0xFFFFFF08`
- ◯ (C) (1) `0x08076000` (2) `0xFFFFFF16`
- ◯ (D) (1) `0x08076008` (2) `0xFFFFFF08`

For the next two subparts only, assume that the code section is not randomized in ASLR (i.e. the addresses given in the assembly printout do not change between executions).

Q4.3 (1 point) What is the value stored in RIP of `stack_editor` if the program is paused at Line 2?

- ◯ (A) `0x08076038`
- ◯ (B) `0x0807603C`
- ◯ (C) `0x08076040`
- ◯ (D) `0x08076044`

Q4.4 (1 point) What is the address of the `call system` instruction within the assembly code for `main`?

- ◯ (A) `0x08076030`
- ◯ (B) `0x08076034`
- ◯ (C) `0x08076038`
- ◯ (D) `0x0807603c`

Q4.5 (8 points) Provide an input **of exactly 8 characters** to the `fgets` on Line 6 that causes `system("sh")` to execute.

Pick **one character** (C, V, D, or S) from each row. For example, to input `CCCCDDVV`, chose "C" for the first four rows, then "D" for the next two rows, and then "V" for the last two rows.

HINT: Your post-exploit stack should look similar to a ret2libc exploit stack. Note that unlike the ret2libc as shown in lecture, we do not need to place 4 bytes of garbage below our argument to `system` *(why might this be?).*

☐ (A) Select this box to get 1 point and void your attempt at this subpart.

○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S
○ (A) C	○ (B) V	○ (C) D	○ (D) S

Q4.6 (2 points) The hint for the previous subpart specified that, unlike in the ret2libc shown in lecture, we do not need to place 4 garbage bytes below our argument to `system`. Which option best explains why this is the case?

○ (A) The argument to `stack_editor` effectively functions as the four bytes of garbage.

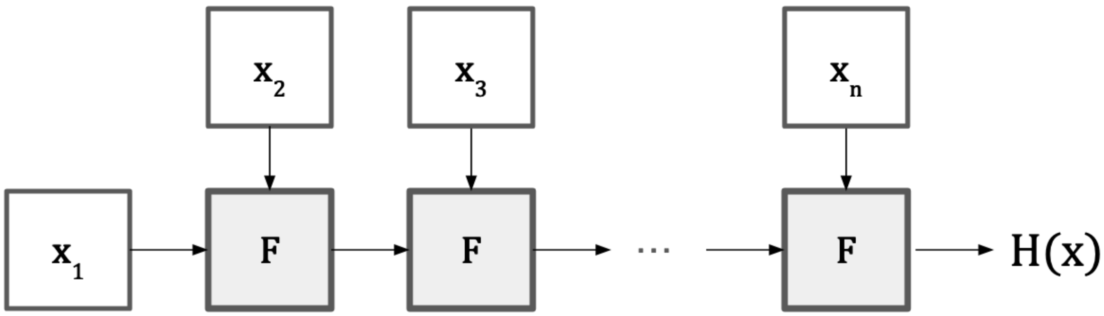
○ (B) The `call` instruction pushes the RIP of `system` onto the stack before moving the EIP.

○ (C) The exploit is not ret2libc, but rather a ret2ret into the address of `system`.

○ (D) The `sh_str` variable is already on the stack and doesn't need to be placed by the exploit.

Q5 *Collision Resistance at a Cheap Price!? Satisfactory* (15 points)

Consider a collision-resistant **compression function** F that takes in two 128-bit inputs and returns a 128-bit output. We use F to build a cryptographic hash function $H(x)$, as shown below:



EvanBot wants to hash arbitrary-length input x. To compute $H(x)$, EvanBot first splits x into n 128-bit blocks, and computes

$$H(x) = F(x_n, F(x_{n-1}, \cdots F(x_3, F(x_2, x_1))))$$

Assume x is always at least two blocks long and an exact multiple of the block length unless otherwise stated.

Q5.1 (2 points) Given hash output $h = H(x_1 \| x_2)$, perform a length-extension attack by giving an expression for $H(x_1 \| x_2 \| y)$, where y is a one-block value chosen by the attacker.

Your expression can include y, F, h, and elementary functions such as $\oplus$, but cannot include x_1 or x_2.

Q5.2 (1 point) Is the MAC construction $\mathsf{MAC}(K, M) = H(K \| M)$ EU-CMA (also known as EU-CPA) secure?

- (A) Yes, because H could still be collision-resistant despite being vulnerable to length-extension attacks.
- (B) Yes, because the adversary does not know K and cannot perform the length-extension attack.
- (C) No, because the adversary can use the length-extension attack to forge MACs for some $M' \neq M$ given $\mathsf{MAC}(K, M)$.
- (D) No, because H's vulnerability to length-extension attacks implies it is not collision-resistant.

Q5.3 (2 points) Suppose for this subpart only that the input x is not necessarily a multiple of the block length and may need padding.

Which padding schemes allow an attacker to find a collision, i.e. $x \neq y$ such that $H(x) = H(y)$? Select all that apply.

Note: $\mathsf{len}(x)$ returns the size of x in bits.

☐ (A) Pad x with 0s until $\mathsf{len}(x)$ reaches a multiple of 128 bits.

☐ (B) Pad x with 0s until $\mathsf{len}(x)$ reaches a multiple of 128 bits, and then add a new block x_{n+1} of all 1s.

☐ (C) If $\mathsf{len}(x)$ is not a multiple of 128, pad x with a single 1 and then 0s until it is a multiple of 128 bits. Otherwise, do nothing.

☐ (D) None of the above.

The rest of this question is independent of the previous subparts.

We're now going to explore insecure candidates for the compression function F. **For each remaining subpart,** give a collision pair $(x_1, x_2), (y_1, y_2)$ such that $F(x_1, x_2) = F(y_1, y_2)$ and $x_1 \| x_2 \neq y_1 \| y_2$.

For example, if $F(a, b) = a$, then a valid solution is $(x_1, x_2) = (1, 0)$, $(y_1, y_2) = (1, 1)$.

Assumptions:

- x_1, x_2, y_1, y_2 must be exactly 128 bits each, but you may answer with a simple integer and assume it is converted to the associated bitstring.
- There may be multiple correct answers. In the example above, $(x_1, x_2) = (5, 7), (y_1, y_2) = (5, 8)$ would also be correct.
- You can use AES encryption E and AES decryption D in your expressions. For example, you can write $\mathsf{E}_3(6)$ or $\mathsf{D}_3(6)$.

HINT: One strategy is to set fixed values for x_1, x_2, y_1 (e.g. $x_1 = 5, x_2 = 6, y_1 = 7$), write $F(x_1, x_2) = F(y_1, y_2)$, and solve for y_2.

Q5.4 (1 point) $F(a, b) = a \oplus b$

☐ (A) Select this box to get 0.25 points and void your attempt at this subpart.

x_1: ______ x_2: ______

y_1: ______ y_2: ______

Q5.5 (2 points) $F(a,b) = \mathsf{E}_a(b)$

☐ (A) Select this box to get 0.25 points and void your attempt at this subpart.

x_1:

x_2:

y_1:

y_2:

Q5.6 (2 points) $F(a,b) = \mathsf{E}_a(b) \oplus a$

☐ (A) Select this box to get 0.25 points and void your attempt at this subpart.

x_1:

x_2:

y_1:

y_2:

Q5.7 (2 points) $F(a,b) = a^b \bmod p$, where p is a large, public cryptographic prime. Assume that a, b are converted from bitstrings to 128-bit unsigned integers during evaluation.

☐ (A) Select this box to get 0.25 points and void your attempt at this subpart.

x_1:

x_2:

y_1:

y_2:

Q5.8 (3 points) $F(a,b) = \mathsf{E}_K(a)[{:}64] \parallel \mathsf{E}_K(b)[{:}64]$, where K is a fixed public constant (i.e. you can use K in your expressions).

Note that [:64] refers to taking the first 64 bits of that value.

☐ (A) Select this box to get 0.25 points and void your attempt at this subpart.

x_1:

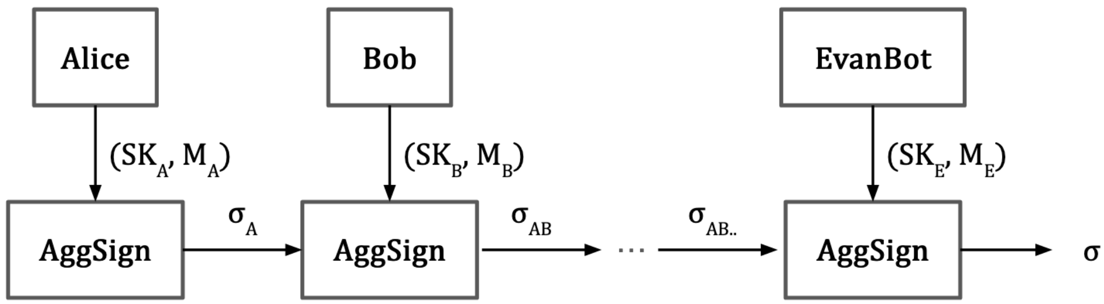
x_2:

y_1:

y_2:

Q6 *Awesome Aggregation - Digital Signatures* (12 points)

Evanbot creates a **sequential aggregate signature scheme**, which enables a group of users to sequentially sign a message list.



For example, Alice starts a petition for CS161 to be a mandatory requirement. Alice signs a *message list* $[M_A]$ with her private key SK_A and produces an aggregate signature σ_A.

Bob now wants to add his name to the petition by creating a signature σ_{AB} on the message list $[M_A, M_B]$. To do so, Bob runs AggSign:

$$\sigma_{AB} = \mathsf{AggSign}(SK_B, M_B, \sigma_A, [PK_A], [M_A])$$

which first verifies the existing signature σ_A with the current message list $[M_A]$, and then creates a new aggregate signature over $[M_A, M_B]$. Verifiers can then use σ_{AB} to verify that Bob signed $[M_A, M_B]$ and that Alice signed $[M_A]$.

The scheme is secure if an adversary cannot forge signatures that are not trivial extensions of existing signatures (a trivial extension would be creating new signatures by running `AggSign` on existing signatures).

Q6.1 (0.5 point) Let σ be an aggregate signature over the message list $[X, Y, Z]$ with public keys PK_A, PK_B, and PK_C (Alice, Bob, Charlie), respectively.

TRUE or FALSE: Given σ, a verifier can conclude that Alice endorses the message Z (i.e. that Alice actively decided to sign a list including Z).

○ (A) TRUE ○ (B) FALSE

Q6.2 (0.5 point) TRUE or FALSE: Given the same σ, a verifier can conclude that Bob endorses the message X (i.e. that Bob actively decided to sign a list including X).

○ (A) TRUE ○ (B) FALSE

The next two subparts are independent from the rest of the question.

Q6.3 (2 points) Consider basic RSA signatures, with $PK = (e, N)$, $SK = d$, and $\mathsf{Sign}(SK, M) = S \equiv M^d \bmod N$. Select the verifying expression.

NOTE: $X \stackrel{?}{\equiv} Y \bmod N$ *returns* `true` *if* $X \equiv Y \bmod N$*, otherwise* `false`*.*

○ (A) $S^d \stackrel{?}{\equiv} M \bmod N$ ○ (C) $S^e \stackrel{?}{\equiv} M \bmod N$

○ (B) $M^e \stackrel{?}{\equiv} S \bmod N$ ○ (D) $M^d \stackrel{?}{\equiv} S \bmod N$

Q6.4 (3 points) Is the RSA signature scheme from the previous subpart EU-CMA (also known as EU-CPA) secure?

○ (A) Yes ○ (B) No

If you selected "No", give a message/signature pair (M, S) with $1 < M < N - 1$ such that S is a valid signature for M without using the private key d.

$M =$ [ ] $S =$ [ ]

Now we will construct sequential aggregate signatures using hash-based RSA signatures. Each user has an RSA keypair with secret key d_i and public key $PK_i = (e_i, N_i)$. Assume that $N_i > N_{i-1}$ for $i > 1$.

For the rest of this question, let $h_k = H([PK_1, \ldots, PK_k], [M_1, \ldots, M_k])$ for brevity.

`AggSign`$(d_k, M_k, \sigma, [PK_1, \ldots, PK_{k-1}], [M_1, \ldots, M_{k-1}])$:

1. Verify that `AggVerify`$(\sigma, [PK_1, \ldots, PK_{k-1}], [M_1, \ldots, M_{k-1}])$ = `true`
2. Return $(\sigma + H([PK_1, \ldots, PK_k], [M_1, \ldots, M_k]))^{d_k} \equiv (\sigma + h_k)^{d_k} \bmod N_k$

`AggVerify`$(\sigma, [PK_1, \ldots, PK_k], [M_1, \ldots, M_k])$:

1. Evaluate $T =$ [ANSWER TO Q6.5]
2. Let $\sigma' = T -$ [ANSWER TO Q6.6] $\bmod N_k$
3. Return `AggVerify`$(\sigma', [PK_1, \ldots, PK_{k-1}], [M_1, \ldots, M_{k-1}])$

The base case of a single-entry list is signed ($H(PK_1, M_1)^{d_1} \equiv h_1^{d_1} \bmod N_1$) and verified as a normal hash-based RSA signature.

Fill in the blanks of the `AggVerify` algorithm.

Q6.5 (2 points)

○ (A) $\sigma^{e_k} \bmod N_k$ ○ (C) $\sigma - h_k \bmod N_k$

○ (B) $\sigma^{d_k} \bmod N_k$ ○ (D) $(\sigma - h_k)^{e_k} \bmod N_k$

Q6.6 (1 point)

○ (A) $h_k^{e_k}$ ○ (C) h_k^{-1}

○ (B) $\sigma - h_k$ ○ (D) h_k

Q6.7 (3 points) Which option best explains why `AggVerify` is secure?

○ (A) Only those with access to the k-th private key d_k can verify their corresponding step.

○ (B) If any `AggSign` in the recursive chain was invalid, then the next modulus N_{k-1} will be malformed.

○ (C) Basic RSA signatures aren't malleable (e.g. you can't derive $\mathsf{Sign}(d, M^2)$ from $\mathsf{Sign}(d, M)$).

○ (D) If any `AggSign` in the recursive chain was invalid, then the corresponding value for σ' as derived in Step 2 of `AggVerify` will be garbage.

Q7 *SQL < PrQL* **(10 points)**

EvanBot has created a concert ticketing app called Boxapp, stored at `boxapp.cs161.org`. Each user has a seat number for one or more concert(s) they are attending.

To find their seat number for a selected concert, a user visits `boxapp.cs161.org/search?q=____`, replacing the blank with the concert name. Boxapp then places the un-sanitized search query on the page (e.g. "You searched for: ___"), followed by the user's seat number for that concert.

The website uses session tokens to authenticate users. Session tokens are stored as cookies with `Domain=cs161.org, Path=/, HttpOnly=False, Secure=True`.

Q7.1 (2 points) Mallory is an on-path attacker. Which actions (by themselves) would allow Mallory to learn the value of a logged-in user's session token? Select all that apply.

☐ (A) The user loads Mallory's site at `https://mallory.org`.

☐ (B) The user loads Mallory's site at `https://mallory.cs161.org`.

☐ (C) The user loads Mallory's site at `https://boxapp.cs161.org/mallory/custom_server`.

☐ (D) The user loads `http://boxapp.cs161.org`.

☐ (E) The user loads `https://boxapp.cs161.org`.

☐ (F) None of the above

Q7.2 (2 points) Mallory wants to steal a user's session token using reflected XSS. Construct a URL that sends the session token to `mallory.org` when a user clicks on the URL. You may use the `post(url, payload)` JavaScript function to send POST requests.

Boxapp uses the two SQL tables shown below:

```
CREATE TABLE sessions (
    username String,
    token String
);
```

```
CREATE TABLE userdata (
    username String,
    concert String,
    seatno String
);
```

When a logged-in user performs a search, the server executes the following two SQL queries:

1. `SELECT username FROM sessions WHERE token = '$token';`
   where `$token` is the user's session token.
2. `SELECT seatno FROM userdata WHERE username = '$result' AND concert='$query';`
   where `$result` is the username from the first query, and `$query` is the user's search query.

Q7.3 (1 point) Select all values for `$query` that would cause the server to returns all `seatno` entries from `userdata`.

Reminder: x AND y OR z = (x AND y) OR z in SQL.

- ☐ (A) `' OR 1=1; --`
- ☐ (B) `' AND username='';--`
- ☐ (C) `'; --`
- ☐ (D) None of the above

Q7.4 (2 points) Mallory now wants to inject a value for Alice's session token, such that the server will return Bob's data whenever Alice uses the search function. Bob is not logged in.

Give an value for Alice's session token, such that for any search, the server returns `seatno` entries for username `'bob'`.

Q7.5 (3 points) EvanBot writes code for deleting a user, and wants to parameterize the SQL query. However, the server is written in Go, and EvanBot only knows how to do parameterized SQL in Python. EvanBot decides to invoke the Python code in the Go code using an `eval_python` function:

```
eval_python( // A function in Go.

  // A raw string passed to the Python interpreter.
  "safeSQL('DELETE FROM userdata WHERE username = ?', ['x'])"

)
```

where `x` is provided by the user and substituted into the raw string before calling `eval_python`.

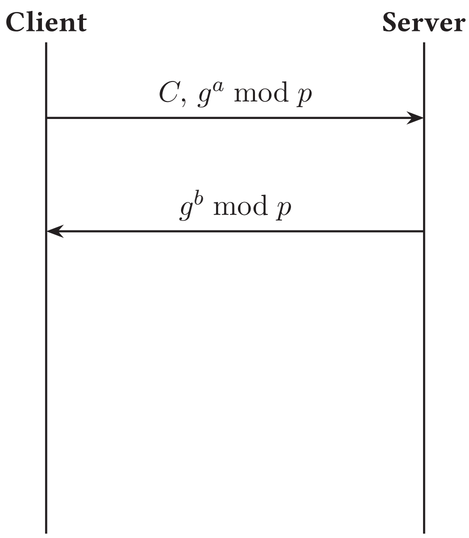
SQL injection is no longer possible, but another attack is possible. In 10 words or fewer, briefly describe or name the attack.

Q8 *Pon de Replay – TLS* **(10 points)**

EvanBot wants to design a new TLS handshake (completely replacing the standard TLS handshake).

For this question, a replay attack from server to client means:

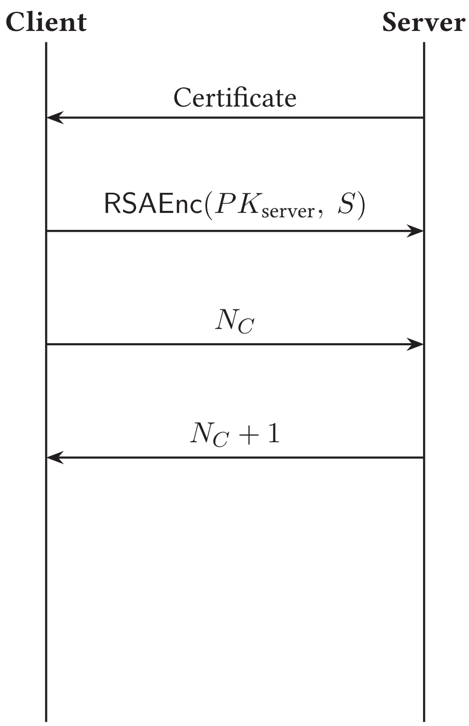
- A MITM attacker records all server-to-client messages (handshake and data) in a connection.
- Later, a client initiates a new connection and the attacker replays all the recorded server-to-client messages, with no modifications. (The attacker blocks all legitimate server messages.)
- The attack succeeds if the client accepts the replayed data.
- A replay attack from client to server is the same with roles swapped, i.e. an attacker replays a client transcript to the server.



1. Client generates a random 128-bit C and a random Diffie-Hellman secret a, and sends C and $g^a \bmod p$.
2. Server generates a random Diffie-Hellman secret b and sends $g^b \bmod p$.
3. Both parties compute $\text{SessionKey} = H\left(g^{ab} \bmod p \,\|\, C\right)$. SessionKey is used to derive the symmetric keys for the session.
4. After the connection ends, the client deletes a, the server deletes b, and both delete SessionKey.

Q8.1 (5 points) Select all true statements about this scheme.

- ☐ (A) A MITM adversary can perform a replay attack from server to client.
- ☐ (B) A MITM adversary can perform a replay attack from client to server.
- ☐ (C) A passive eavesdropper can read encrypted data sent after the handshake completes.
- ☐ (D) A MITM can tamper with the handshake to read and modify encrypted data in both directions.
- ☐ (E) This scheme has forward secrecy.
- ☐ (F) None of the above.



1. Server sends a certificate for its long-term RSA public key, PK_{server}. Client verifies this certificate.
2. Client generates a random 128-bit S and sends $\mathsf{RSAEnc}(PK_{\text{server}},\ S)$.
3. Client generates and sends a random 128-bit nonce N_C.
4. Server sends $N_C + 1$. Client verifies that this value is one more than the client's generated N_C.
5. Both parties compute $\text{SessionKey} = H(S \parallel N_C)$. SessionKey is used to derive the symmetric keys for the session.
6. After the connection ends, both parties delete S and SessionKey.

Q8.2 (5 points) Select all true statements about this scheme.

- ☐ (A) A MITM adversary can perform a replay attack from server to client.
- ☐ (B) A MITM adversary can perform a replay attack from client to server.
- ☐ (C) A passive eavesdropper can read encrypted data sent after the handshake completes.
- ☐ (D) A MITM can tamper with the handshake to read and modify encrypted data in both directions.
- ☐ (E) This scheme has forward secrecy.
- ☐ (F) None of the above.

Q9 *ARP, it's in the game!* (7 points)

Q9.1 (1 point) For this subpart only, suppose we change ARP requests to include a 128-bit random number. The sender only accepts an ARP response if the response includes the number from the request.

Consider an on-path attacker that can send at most 200 spoofed responses before the legitimate response arrives. Is this modified ARP scheme secure against ARP spoofing?

- ○ (A) Yes, because the attacker cannot guess the random number with non-negligible probability.
- ○ (B) Yes, because the attacker does not know where to send the spoofed ARP response.
- ○ (C) No, because the attacker can see the original ARP request and learn the random number.
- ○ (D) No, because the attacker can guess the random number with non-negligible probability.

Q9.2 (1 point) Suppose a user is the victim of an ARP spoofing attack by an on-path attacker. Select all true statements.

- ☐ (A) The attacker can eavesdrop on the user's TLS connections.
- ☐ (B) The attacker can become a MITM for the user's HTTP connections.
- ☐ (C) The attacker can spoof valid DNSSEC responses.
- ☐ (D) None of the above.

Q9.3 (1 point) Which fields are included in a DHCP offer from the router? Select all that apply.

- ☐ (A) User's assigned IP address
- ☐ (B) User's assigned MAC address
- ☐ (C) Router's IP address
- ☐ (D) DNS server's IP address
- ☐ (E) DNS server's MAC address
- ☐ (F) None of the above

Q9.4 (2 points) Is it true that user requests over UDP are more vulnerable to spoofing attacks from off-path attackers than user requests over TCP?

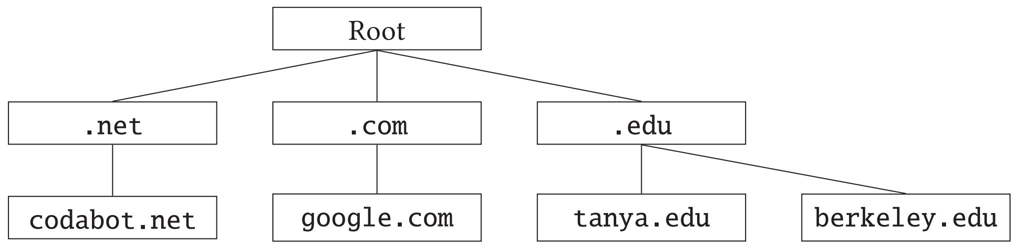
- ○ (A) Yes, because an off-path attacker needs to guess fewer fields to spoof a UDP packet.
- ○ (B) Yes, because TCP is a best-effort protocol unlike UDP.
- ○ (C) No, because UDP's simple checksum prevents creation of valid spoofed packets.
- ○ (D) No, because UDP's unreliable delivery means spoofed packets are likely to be discarded.

Q9.5 (2 points) Does TCP provide confidentiality? Select the best option.

- ○ (A) Yes, because TCP's three-way handshake encrypts the data stream.
- ○ (B) Yes, because TCP's sequence numbers ensure that only the recipient can read the data.
- ○ (C) No, because TCP's checksum mechanism is not a secure MAC.
- ○ (D) No, because TCP does not encrypt its payload.

Q10 * *Despite everything, it's still DNS* **(11 points)**

Jonah wants to learn some IP addresses using DNS. For this question, no zones exist besides the ones in the diagram below.



Q10.1 (1 point) Assuming the DNS cache begins empty, how many DNS requests does the recursive resolver need to send to learn the IP address of `evanbot.tanya.edu`?

○ (A) 1 ○ (B) 2 ○ (C) 3 ○ (D) 4

Q10.2 (1 point) Assuming all records from the previous subpart remain in the cache, how many DNS requests does the recursive resolver need to send to learn the IP address of `cookies.tanya.edu`?

○ (A) 1 ○ (B) 2 ○ (C) 3 ○ (D) 4

The name server for `tanya.edu` has been hacked by an attacker. They create a malicious `A` record mapping `eecs.berkeley.edu` to their IP of `161.0.0.1`. The attacker then adds this `A` record to the Additional section of every reply from the `tanya.edu` name server.

For all remaining subparts, assume that **bailiwick checking is enabled**, and the DNS cache starts empty each time. Each subpart is independent (i.e. they all start with an empty cache).

Q10.3 (1 point) If Jonah's recursive resolver performs a DNS lookup for `www.codabot.net`, will the resolver's cache contain an entry for `eecs.berkeley.edu`?

○ (A) Yes ○ (B) No

Q10.4 (1 point) If Jonah's recursive resolver performs a DNS lookup for `evanbot.tanya.edu`, will the resolver's cache contain an entry for `eecs.berkeley.edu`?

○ (A) Yes ○ (B) No

Q10.5 (1 point) If Jonah's resolver implements source port randomization, does the attacker need to guess the randomized port number in their response?

○ (A) Yes ○ (B) No

Suppose that the hacked `tanya.edu` nameserver now replies to requests for `evanbot.tanya.edu` with an `A` record containing the attacker's IP `161.0.0.1`.

Q10.6 (1 point) True or False: If Jonah's resolver performs a DNSSEC lookup for `evanbot.tanya.edu`, his resolver will cache that `evanbot.tanya.edu`'s IP address is `161.0.0.1`.

Assume the attacker has access to the hacked name server's keys, and the hacked name server is still endorsed by the `.edu` name server.

○ (A) True ○ (B) False

The following two subparts are independent of all previous subparts.

An off-path attacker is performing a Kaminsky attack and can send n fake responses for each DNS request before the legitimate response arrives. Assume source port randomization is disabled and that negative answers (domain does not exist) are cached.

Q10.7 (3 points) In this subpart, the user loads `fake.google.com` only once.

What is the approximate probability that the attacker succeeds in poisoning the IP address of `www.google.com`?

- ○ (A) $\frac{n}{2^{16}}$
- ○ (B) $\frac{n}{2^{32}}$
- ○ (C) $\frac{n}{2^{64}}$
- ○ (D) 1

Q10.8 (2 points) In this subpart, the user loads `fake.google.com` 200 times, one after the other.

True or False: Compared to the previous subpart, the attacker's probability of success for the same cache poisoning attack is strictly greater.

- ○ (A) True
- ○ (B) False

Everything below this line will not be graded.

Post-Exam Activity: Hat

Comment Box

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

If you feel like there was an ambiguity in the exam, please put it in the box above. 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.