1. (21 points) Circle True or False. Do not justify your answer.

   (a) True or [✓ False]: DHCP spoofing can be prevented by using Ethernet instead of wireless.

   Solution: DHCP requests and responses are broadcast.

   (b) [✓ True] or False: SYN Cookies help defend against distributed syn flooding attacks.

   (c) [✓ True] or False: Source port randomization is a helpful defense against Kaminsky blind spoofing.

   (d) True or [✓ False]: An off-path attacker is more powerful than an on-path attacker: anything an on-path attacker can do, so can an off-path attacker.

   (e) [✓ True] or False: An on-path attacker is more powerful than an off-path attacker: anything an off-path attacker can do, so can an on-path attacker.

   (f) True or [✓ False]: Internet censorship requires an in-path attacker (i.e., an on-path attacker that can both observe all packets and also drop any packets the censor wishes).

   Solution: As we saw in the lecture on Internet censorship, an on-path attacker (who can observe all packets but not drop packets) can censor connections.

   (g) [✓ True] or False: If you use HTTPS but not DNSSEC, the confidentiality of data you send over HTTPS is protected against on-path attackers (ignoring implementation bugs and/or CA failures).

   Solution: Without DNSSEC, you might be connecting to the wrong server, but even if you're connecting to the attacker, the TLS connection will fail because the attacker doesn't have the right private key or cert to impersonate the real server.

   (h) True or [✓ False]: If you use HTTP and DNSSEC, the confidentiality of data you send over HTTP is protected against on-path attackers (ignoring implementation bugs and/or CA failures).

   Solution: DNSSEC only ensures that you're sending packets to the correct IP address. On-path attackers can observe the packets you send directly, even if you send them to the correct server.

   (i) True or [✓ False]: CBC mode encryption provides both confidentiality and integrity.

   Solution: It doesn't provide integrity.

(j) $\boxed{\text{TRUE}}$ or FALSE: CBC mode encryption provides confidentiality against chosen-plaintext attacks (IND-CPA security).

> **Solution:** See lecture notes.

(k) TRUE or $\boxed{\text{FALSE}}$: Using a pseudorandom number generator, seeded by the current time of day (measured to microsecond precision), is a good way to generate an AES key.

> **Solution:** Such a seed is guessable, enabling brute-force attacks to recover the AES key.

(l) $\boxed{\text{TRUE}}$ or FALSE: Cryptography is a reasonable defense if an adversary might be able to eavesdrop on network packets.

(m) TRUE or $\boxed{\text{FALSE}}$: Stack canaries are a good defense against heap-based buffer overflows.

> **Solution:** They're aimed at overflows of stack-allocated buffers; an overflow of a buffer in the heap won't overwrite the canary and won't be detected by the stack canary defense.

(n) TRUE or $\boxed{\text{FALSE}}$: Fuzz-testing requires access to source code to find vulnerabilities.

> **Solution:** Fuzz-testing doesn't require access to source code; you can fuzz test a binary.

(o) $\boxed{\text{TRUE}}$ or FALSE: Prepared statements are a good defense against SQL injection.

(p) TRUE or $\boxed{\text{FALSE}}$: Setting the "secure" flag on a cookie (so it will only be sent over HTTPS) is a good defense against CSRF.

(q) TRUE or $\boxed{\text{FALSE}}$: TLS does not provide confidentiality if you use a TCP implementation whose TCP initial sequence numbers are predictable.

(r) TRUE or $\boxed{\text{FALSE}}$: Access control ensures that authorized users who have access to sensitive data won't misuse it.

> **Solution:** Access control only limits which users can access the data; it can't restrict what they do with the data, once they have it.

(s) TRUE or $\boxed{\text{FALSE}}$: Alice wants to communicate with Bob using Tor with 3 intermediaries. If the first 2 intermediaries are dishonest, they will be able to determine that Alice and Bob are communicating.

Solution: They can see packets coming from Alice, but can't tell where they are going (they can see that they're going towards the 3rd intermediary, but can't tell where they'll go from there).

(t) TRUE or $\boxed{\text{FALSE}}$: When a client sends a message over the Tor network, Tor's onion routing works because each intermediary encrypts the message they receive with the public key of the following intermediary, so no one else can decrypt the messages.

Solution: All encryption is done by the client, not by the intermediaries. The client encrypts multiple times, in a nested fashion, once per intermediary; each intermediary then removes one level of encryption.

(u) $\boxed{\text{TRUE}}$ or FALSE: The homomorphic properties of encryption allow servers to compute products or sums of encrypted data without having to decrypt it, as long as the data is encrypted with the right encryption algorithm.

Solution: e.g., RSA or Paillier.

2. (12 points) You are doing a security test of a website, `ShopSMart.com`. In each part below, based on the symptoms you observe, give the name of the type of vulnerability that is most likely responsible for those symptoms. Write just the name of the type/category of vulnerability (e.g., "buffer overrun"); you don't need to write a detailed description of the specific attack or vulnerability.

(a) You try to sign up for an account and type `Can't stop the signal` into the text field for entering your last name. When you click the button to submit the form, you see an error page that mentions a syntax error in the database query.

What type of vulnerability does this most likely indicate?

Solution: SQL injection

(b) You sign up under your real name, then place an order for 65539 bookmarks: you add the bookmark to your cart and enter a quantity of 65539. You place the order. A week later, a giant package arrives at your doorstep containing 65539 bookmarks. You check your credit card bill, and you see you've only been charged for 3 bookmarks.

What type of vulnerability does this most likely indicate?

Solution: Integer overflow or integer conversion vulnerability (other answers may be valid as well).

(c) `ShopSMart.com` has a one-click buy feature: on the web page for each item, there is a button labelled "Buy instantly!". If you click that button while logged in, it instantly places an order for the item and charges your credit card on file, without requiring you to go through any other web pages (there's no separate checkout page, no confirmation page—the order is placed immediately). You view the HTML source for the page for the bookmark, and you see

```
<FORM ACTION="http://shopsmart.com/instantbuy&item=bookmark">
  <BUTTON TYPE="submit" VALUE="Buy instantly!">
</FORM>
```

You know this means that when the button is clicked, the browser will load the page `http://shopsmart.com/instantbuy&item=bookmark`. Looking through the rest of the page source, the page doesn't appear to contain any Javascript.

What type of vulnerability does this most likely indicate?

Solution: CSRF.

3. (8 points) You have a secure cryptographic hash function $H : \{0,1\}^{2n} \to \{0,1\}^n$ that takes $2n$-bit inputs and outputs a n-bit digest. You want to create a secure cryptographic hash function $F : \{0,1\}^{4n} \to \{0,1\}^n$ that takes $4n$-bit inputs and outputs a n-bit digest. How could you do this? Fill in the definition below, assuming w, x, y, z are each n-bit values:

Solution: $F(w,x,y,z) = H(H(w,x), H(y,z)$ or $F(w,x,y,z) = H(w, H(x, H(y,z)))$ or $F(w,x,y,z) = H(H(H(w,x),y),z)$, etc.

Also fine: $F(w||x||y||z) = H(H(w||x)||H(y||z))$, etc.

(Don't explain or justify your answer.)

4. (12 points) (a) A *code injection* attack is one that involves attacker-chosen code being introduced into an application and then executed. Which of the following attacks are a form of code injection attack? Circle all that apply.

   1. [✓ Buffer overflow]
   2. Port scanning
   3. Kaminsky-style blind spoofing of DNS responses
   4. CSRF
   5. TCP SYN flooding
   6. Clickjacking
   7. [✓ XSS]
   8. Browser-in-browser attacks
   9. None of the above

(b) In iOS, each app runs with the userid `mobile`. In Android, each app runs with a unique userid specific to that app. Which of these two better follows the principle of least privilege? Why?

Solution: Android. Each userid can be assigned a different set of permissions.

5. (12 points) The Chrome browser contains a built-in filter which attempts to block certain attacks. When a user visits `http://website.com/foo.html?name=value`, Chrome looks at the HTML response and checks all Javascript scripts in the page to see if any of them are contained in (that is, a substring of) `value`; if so, Chrome refuses to execute that script.

For each part, circle Yes or No, then explain in a sentence why or why not. Assume the attacker doesn't try to do anything special to bypass the filter.

(a) Will this stop reflected XSS attacks?

[✓ YES] NO

Explanation:

Solution: Yes (at least some): they involve malicious data being copied from `value` into the response.

(b) Will this stop stored XSS attacks? (Stored XSS is also known as persistent XSS.)

YES [✓ NO]

Explanation:

Solution: The malicious data in the HTML response comes from the database, not from a URL parameter.

(A prior request causes malicious data to be stored into the database; then later the victim visits a page that causes it to be copied from the database to the HTML response. That's two requests. Neither one will be blocked by Chrome.)

(c) Will this stop CSRF attacks?

YES [✓ NO]

Explanation:

Solution: In CSRF the harm is done by the server acting on the request; the response is irrelevant.

6. (15 points) (a) When users of `bank.com` are logged in, a request to `bank.com/session.js` returns a Javascript file containing

```
var session_id = "0123456789";
```

except that 0123456789 is replaced with the session ID for the user who made the request.

An attacker controls `evil.com` and would like to learn Alice's session ID for `bank.com`. How can the attacker do this? Explain why the same-origin policy doesn't stop this attack. (Assume the attacker can get Alice to visit `evil.com`.)

Solution: Put `<SCRIPT SRC="http://bank.com/session.js">` into `evil.com`'s page, followed by Javascript that reads the `session_id` variable and sends it back to `evil.com`. Allowed because of the special exception in the same-origin policy for scripts.

(It's fine to assume Alice is logged in for this part; otherwise, no attack is possible.)

(b) When `bank.com` learns of this problem, they fix it by beginning all Javascript files with

```
if (!document.location.includes("http://bank.com")) {
    while (1) {}   // infinite loop
}
```

Explain why this doesn't work. How could an attacker defeat this defense?

Solution: An attacker could put malicious stuff on `http://bank.com.evil.com/`, or on `http://evil.com/foohttp://bank.com/`.

(c) Propose better Javascript code to put at the start of all Javascript files.

Solution:

```
if (!document.location.startsWith("http://bank.com/")) {
    while (1) {}   // infinite loop
}
```

or

```
if (document.domain != "bank.com")) {
    while (1) {}   // infinite loop
}
```

or some similar solution.

We're not testing you on your knowledge of specific Javascript APIs, so it was fine to make up method names if you didn't know what methods to use, as long as your intent is clear.

7. (18 points) Tracy proposes a new mode of operation for AES encryption, which we'll call the Tracy block chaining (TBC) mode. The message M is split into 16-byte blocks, $M =$

$M_1||M_2||\cdots||M_k$. The ciphertext $C = C_0||C_1||\cdots||C_k$ is computed as follows: we pick a random 16-byte IV, let C_0 = this IV, let M_0 = this IV, and define $C_i = E_k(M_{i-1} \oplus M_i)$ for $i = 1, 2, \ldots, k$. Here $E_k(\cdot)$ denotes encryption of a single block using the AES block cipher and $D_k(\cdot)$ denotes decryption of a single block using the AES block cipher.

(a) Define the decryption algorithm. If the ciphertext is $C_0||C_1||\cdots||C_k$, give an equation specifying how the recipient can recover $M_1, \ldots, M_k$ as a function of the C_i's:

Solution: $M_i = D_k(C_i) \oplus M_{i-1}$.
$M_i = D_k(C_i) \oplus D_k(C_{i-1}) \oplus \ldots \oplus D_k(C_1) \oplus C_0$ would also be fine.

(b) Is TBC mode IND-CPA secure? Why or why not? Circle "yes" or "no", then justify your answer.

YES or [✓ NO]

Justification:

Solution: There are many possible justifications. Here are a few examples, any one of which would be sufficient:

- The plaintext $0||Y$ will be distinguishable from $X||Y$ (the ciphertext for the former will have $C_1 = C_2$).
- The plaintext $X||X||X$ is distinguishable from $X||Y||Z$. (The last two blocks of ciphertext will be identical, for the first plaintext.)
- If you send the same message twice, all but the first 32 bytes of the ciphertext will be identical both times.

(c) Alice encrypts the message M = `cs161 is so easy` using TBC. M is 16 bytes, and Alice uses no padding, so here $k = 1$. Let the resulting ciphertext be $C = C_0||C_1$. Alice sends C to Bob.

Mallory is a man in the middle. Mallory wants to tamper with this ciphertext and replace it with some other ciphertext C', so that when Bob decrypts C', he will receive M' = `cs161 is so hard`. Can Mallory do this? Circle "yes" or "no", then justify your answer: if you circled yes, describe the modified ciphertext C' that Mallory should send; if you circled no, explain why Mallory can't do it.

[✓ YES] or NO

Justification:

Solution: $C = C'_0||C_1$ where $C'_0 = C_0 \oplus M \oplus M'$.

8. (20 points) The TLS specification says that, during the handshake, the browser should send a random 256-bit number R_B and the server should send a random 256-bit number

R_S. This question asks about the consequences if a browser or server fails to implement this correctly. Each part is independent. Assume the browser and server use RSA-based key exchange (not Diffie-Hellman).

(a) Suppose Alice downloads a buggy version of the Chrome browser that implements TLS incorrectly. Instead of picking R_B randomly, it increments a counter and sends it instead. Which of the following attacks are possible? Circle all that apply.

1. A man-in-the-middle can compromise Alice's confidentiality (e.g., learn the data she sends over TLS).
2. If Alice visits a HTTPS URL, a man-in-the-middle can successfully replay that HTTP request to the server a second time.
3. If Alice visits the same HTTPS URL twice, when Alice visits that URL the second time a man-in-the-middle can successfully replay the HTML page that was returned by the server on Alice's first visit.
4. A man-in-the-middle can learn the symmetric MAC keys that protect data sent over TLS connections initiated by Alice's browser.
5. [✓ None of the above]

> **Solution:** None will be possible. Even if R_B were to repeat, Alice would choose a different pre-master secret each time and thus they'll end up with different symmetric keys for each different connection, so it's not possible for a MITM to replay an HTML page returned on a prior connection.

(b) Alice downloads a patch for her buggy version of Chrome, which fixes the previous problem but introduces another bug. Now, her client generates the pre-master secret by incrementing a counter. Which of the following attacks are possible? Circle all that apply.

1. [✓ A man-in-the-middle can compromise Alice's confidentiality (e.g., learn the data she sends over TLS).]
2. [✓ If Alice visits a HTTPS URL, a man-in-the-middle can successfully replay that HTTP request to the server a second time.]
3. [✓ If Alice visits the same HTTPS URL twice, when Alice visits that URL the second time a man-in-the-middle can successfully replay the HTML page that was returned by the server on Alice's first visit.]
4. [✓ A man-in-the-middle can learn the symmetric MAC keys that protect data sent over TLS connections initiated by Alice's browser.]
5. None of the above

> **Solution:** The MITM can recover the pre-master secret by brute-forcing it, infer the symmetric keys, and then all security is lost. For example, an attacker

can decrypt all the traffic Alice sent over TLS, and then inject it into a new TLS session from Alice (encrypted and MAC'ed with the proper keys for that connection, which can also be recovered in the same way).

Replay means that the recipient receives the message a second time. (Successfully means that the recipient doesn't reject/detect the message when it is replayed, i.e., that the second copy is accepted as if it were intended to be sent by Alice.)

(c) Alice downloads an updated version of Chrome that is finally correct. With her fixed browser, she visits `https://lazycodrz.com/`. That server's TLS implementation has a bug: instead of picking R_S randomly, it always sends all zeros. Which of the following attacks are possible? Circle all that apply.

1. A man-in-the-middle can compromise Alice's confidentiality (e.g., learn the data she sends over TLS).
2. [✓ If Alice visits a HTTPS URL, a man-in-the-middle can successfully replay that HTTP request to the server a second time.]
3. If Alice visits the same HTTPS URL twice, when Alice visits that URL the second time a man-in-the-middle can successfully replay the HTML page that was returned by the server on Alice's first visit.
4. A man-in-the-middle can learn the symmetric MAC keys that protect data sent over TLS connections initiated by Alice's browser.
5. None of the above

Solution: Attack 2 is possible because the symmetric keys are derived as a function of the pre-master secret, R_B, and R_S. If the MITM replays the entire handshake to the server, the pre-master secret and R_B will automatically be identical; and due to the bug in the server, R_S will be identical as well, so the same symmetric keys will be derived, and encrypted and MAC'ed data from the prior connection can be replayed a second time.

9. (14 points) In older versions of iOS, the implementation of `malloc()` looked approximately like this:

```
struct malloc_header {
    size_t len;
}
void * malloc(size_t len) {
    struct malloc_header *hdr;
    size_t n;
    n = sizeof(malloc_header) + len;
                           // (*)
    hdr = system_alloc(n); // allocate n bytes of memory
    hdr->len = n;
    return hdr + 1;
}
```

You can assume the call to `system_alloc()` will never fail; it always succeeds in allocating `n` bytes of memory. You can also assume this runs on a 32-bit architecture and `size_t` is a 32-bit value.

void free(void *p) struct malloc$_header hdr = p; hdr = hdr - 1; system_free(hdr, hdr-> len)$;

(a) This implementation of `malloc()` could create a security problem for apps. Give an example of an invocation of `malloc()` that could be problematic. Make sure to show a specific value for `malloc()`'s parameter (it's OK to use hex or a mathematical expression). Explain briefly why this causes a problem.

> **Solution:** `malloc(`$2^{32} - 1$`)`
>
> This causes an integer overflow when calculating `n`, so it will allocate many fewer bytes than the app is expecting.
>
> Answers of $2^{32} - 2$ and $2^{32} - 3$ were also valid. $2^{32} - 4$ is valid if you assume that `system_alloc(0)` returns a zero-byte buffer rather than `NULL` (a reasonable assumption).
>
> Some people asked about the type of the argument to `system_alloc()`. Even if the type were `uint64_t` (which seems unlikely, given that we're on a 32-bit system), the code would still be buggy. The integer overflow happens when assigning to `n`, which is a `size_t`, so by the time this is passed to `system_alloc()`, the overflow has already happened and the damage is inevitable.

(b) Fix this problem by adding some extra code at the line marked `(*)`. What C code should we add there?

> **Solution:**
>
> ```
> if (n < sizeof(struct malloc_header))
>     return NULL;
> ```
>
> or
>
> ```
> if (n < len)
>     return NULL;
> ```
>
> An answer like
>
> ```
> if (n < 4)
>     return NULL;
> ```
>
> is not so good. It's better not to hardcode an assumption about the size of a `struct malloc_header`: if the compiler decides to insert padding bytes into

the `struct`, so the `struct` is 8 bytes long, then this code will be insecure.

10. (18 points) After doing Project 2, Neo decided to make a simple version of the Part 1 client that runs in the browser using only JavaScript. Neo calls his new service CryptoCatalog. It gives each registered user a single encrypted text file. The encrypted file is stored on the CryptoCatalog server (which is untrusted, like the storage server in Project 2). There is no sharing or revocation, and there is only one file per user.

    A new user visits CryptoCatalog.com and creates an account (setting a username and password). After logging in, the CryptoCatalog JavaScript code prompts the user for their encryption passphrase (which is *different* from their login password). Using that passphrase, it generates two 128-bit symmetric keys, k_c and k_i, for confidentiality and integrity, by hashing the passphrase 10,000 times with SHA-256: H(H(...H(passphrase)...)) $= k_c \mid\mid k_i$. The JavaScript code gets the encrypted file from the server, and then authenticates and decrypts it using k_i and k_c. Whenever the user saves the file, the new contents are encrypted using AES-CBC under k_c and then the ciphertext is MAC'd using HMAC-SHA256 under k_i.

    (a) Can evil.com read the plaintext file of a user of CryptoCatalog, if the user doesn't visit evil.com? Briefly, why or why not?

    Solution: No, evil.com won't know the keys. Or, No, assuming the user has chosen a good passphrase, as evil.com can't guess the keys.

    (b) Can evil.com read the plaintext file of a user of CryptoCatalog, if the user visits evil.com (but doesn't interact with evil.com at all)? Briefly, why or why not?

    Solution: No, the same-origin policy doesn't give evil.com any special access to CryptoCatalog.com.

    Or, No, assuming the user has chosen a good passphrase, due to same-origin policy.

    (c) Evil.com was able to steal a copy of CryptoCatalog's database of encrypted files. How could evil.com decrypt at least some of the files?

    Solution: Guess passphrases using a dictionary attack with common passwords.

    (d) Suppose evil.com manages to compromise the CryptoCatalog servers and gains complete control over them. How can evil.com break the confidentiality of all of CryptoCatalog's users' files?

Solution: Have CryptoCatalog.com send malicious Javascript to the users. The malicious Javascript could capture the passphrase and send it to evil.com.

(e) With regard to the previous question, why is the web version, CryptoCatalog, more vulnerable than your Project 2 client?

Solution: Because the CryptoCatalog server can change the client Javascript code, whereas in Project 2 the server can't modify your client code.

(Another difference: your Project 2 code used randomly generated symmetric keys, whereas this derives them from a passphrase.)

11. (24 points) You're sitting in a coffee shop enjoying a mocha latte and some relaxing computer security reading online at `http://awesome-security-stuff.com`. You're connected on the coffee shop's wifi network.

(a) Assuming you are only browsing `http://awesome-security-stuff.com`, who is potentially able to observe what articles you are reading? Circle all that apply.

1. [✓ Other coffee shop patrons]
2. [✓ The manager of the store next door to the coffee shop who occasionally leeches off of the coffee shop's wifi]
3. Your friend in a dorm a few miles away
4. [✓ The coffee shop's ISP]
5. [✓ The website `awesome-security-stuff.com`]
6. None of the above

(b) Name two technologies that could reduce the number of parties in part (a) that can observe your traffic. Do not give explanations, simply write down their names.

Solution: Any two of TLS (HTTPS), WPA (encrypted Wifi), VPN, Tor.

(c) If you use both the technologies you listed in part (b), who will still be able to know a complete list of all the articles you view?

1. Other coffee shop patrons
2. The manager of the store next door to the coffee shop who occasionally leeches off of the coffee shop's wifi
3. Your friend in a dorm a few miles away
4. The coffee shop's ISP
5. [✓ The website `awesome-security-stuff.com`]
6. None of the above

Solution: If you wrote Tor in part (b), you should circle item 6 (None of the above), as Tor prevents the website from knowing who is reading each article. Otherwise, you should circle item 5 (the website), as with TLS, WPA, and/or a VPN, `awesome-security-stuff.com` still sees what articles you read.

(d) You notice that each article has a Facebook Like button icon, loaded from `facebook.com`, allowing you to indicate on Facebook that you enjoyed this article. If Facebook wanted to, could it track what articles you are visiting, if you don't click on the Like button? Circle yes or no, then explain why or why not.

[✓ Yes] No

Explanation:

Solution: Your browser loads the icon by making a request to Facebook, and that request will send the cookies for `facebook.com`, so if you've ever logged onto Facebook in the past, Facebook can tell you visited that article.

Some clarifications:

- It's perhaps most obvious that this is possible if you are logged on, so a simple answer is to say, yes, if you're logged on to Facebook.
- Even if you're logged off, Facebook can probably still track what articles you're visiting. If you log on, then log off, your browser will likely still retain cookies for `facebook.com` that allows Facebook to continue to uniquely identify you (and link you to your prior login). Those will be sent with the request to load the Like button icon, so Facebook can identify you at that point—even if you've since logged off. That said, this is probably academic. Based on how Facebook is designed, most Facebook users are probably logged on to Facebook essentially all the time, even if they're not actively using Facebook right then.
- When your browser sends a request to Facebook for the Like icon, the `Referer` header will list which article you're reading, so Facebook can see which article you're reading (from the `Referer`) and identify you (from the cookies sent with the request).

(e) The coffee shop only allows you to surf online for 30 minutes, but you usually like to spend more than that there.

After 30 minutes, you notice that visiting any URL in the browser results in a blocked page (indicating you can no longer browse the Internet). However, you notice you are able to successfully ping any valid domain name. Describe how you might prepare to surf for longer in the future.

Solution: Set up a server to help you tunnel IP over DNS.
Or, ICMP tunneling (since you can ping).

12. (26 points) Sarah is using a laptop on her home network to browse the Internet over HTTP. Consider an adversary with a smartphone and an Internet connection and the following capabilities:

    A. No additional capabilities
    B. A $10k GPU compute cluster
    C. Can compromise Sarah's home router
    D. Can compromise the primary nameserver for the zone
    E. Can compromise the primary nameserver and offline signing key for the zone

    (a) In the following table, **mark with an X** which attacks each adversary could successfully perform against Sarah if everyone uses **DNS**: without DNSSEC:

	A	B	C	D
Spoof existence of records that actually don't exist in the zone				
Spoof values of records that exist in the zone				
Spoof non-existence of records that actually do exist in the zone				
Enumerate all names in the zone				

Answer: A B C D

Spoof1 X X Spoof2 X X Spoof3 X X Enum

	A	B	C	D
Spoof existence of records that actually don't exist in the zone			X	X
Spoof values of records that exist in the zone			X	X
Spoof non-existence of records that actually do exist in the zone			X	X
Enumerate all names in the zone				X

    (b) Remember that when using DNSSEC, the zone is signed with a key that is not stored on the primary nameserver. Instead, the key is stored at an "offline" location. Also, NSEC records sign each adjacent pair of names in the zone. In the following table, **mark with an X** which attacks each adversary could successfully perform against **DNSSEC with NSEC records**:

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					
Spoof values of records that exist in the zone					
Spoof non-existence of records that actually do exist in the zone					
Enumerate all names in the zone					

Answer: A B

C D E Spoof1 X Spoof2 X Spoof3 X Enum X X X X X

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					X
Spoof values of records that exist in the zone					X
Spoof non-existence of records that actually do exist in the zone					X
Enumerate all names in the zone	X	X	X	X	X

(c) NSEC3 records were introduced to fix certain issues with NSEC records: each name in the zone is hashed, the hashes are sorted, and each NSEC3 record signs an adjacent pair of hash digests. In the following table, **mark with an X** which attacks each adversary could successfully perform against **DNSSEC with NSEC3 records**:

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					
Spoof values of records that exist in the zone					
Spoof non-existence of records that actually do exist in the zone					
Enumerate all names in the zone					

Answer: A B

C D E Spoof1 X Spoof2 X Spoof3 X Enum X X X

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					X
Spoof values of records that exist in the zone					X
Spoof non-existence of records that actually do exist in the zone					X
Enumerate all names in the zone		X		X	X

(d) NSEC5 records are a further improvement. They work as follows.

When signing a zone, for each name n in the zone, compute $R(n) = H(Sign_K(n))$ where K is a secondary RSA private key and H is a secure cryptographic hash function. The secondary signing key K is separate from the main key used to sign the zone. Sort all the R values in the zone and sign each consecutive pair (R_i, R_{i+1}) using the primary signing key. The signed zone and the secondary signing key are stored on the primary nameserver.

When answering a query for a non-existent name n, the nameserver responds with $Sign_K(n)$ (generated on-the-fly) and the NSEC5 record that represents the gap containing $R(n)$.

In the following table, **mark with an X** which attacks each adversary could successfully perform against **DNSSEC with NSEC5 records**:

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					
Spoof values of records that exist in the zone					
Spoof non-existence of records that actually do exist in the zone					
Enumerate all names in the zone					

Answer: A B

C D E Spoof1 X Spoof2 X Spoof3 X Enum X X

	A	B	C	D	E
Spoof existence of records that actually don't exist in the zone					X
Spoof values of records that exist in the zone					X
Spoof non-existence of records that actually do exist in the zone					X
Enumerate all names in the zone				X	X