Paxson Spring 2011 — CS 161 Computer Security — Final Exam

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

SIGN your name: _____

PRINT your class account login: `cs161-`_____

Your TA's name: _____

Your section time: _____

Name of the person sitting to your left: _____ Name of the person sitting to your right: _____

You may consult two sheets of notes (each double-sided). You may not consult other notes, textbooks, etc. Calculators and computers are not permitted. Please write your answers in the spaces provided in the test. We will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

You have 170 minutes. There are 9 questions, of varying credit (300 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

Do not turn this page until your instructor tells you to do so.

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	36	24	38	35	24	36	32	35	40	300
Score:										

Problem 1 ***Software Vulnerabilities*** **(36 points)**

A broad class of security vulnerabilities come from attackers exploiting *data vs. instructions*. In these attacks, attackers provide input that's intended to be used solely as data (operated on in a pre-determined manner), but due to a flaw in the processing of that data, elements of the data instead become treated as instructions (programming) that control execution. That is, some of the data provided by the attacker winds up being interpreted as "code" of some sort, even though that was not the intent of the programmer.

For each of the following types of attacks, circle **YES** if it exploits (or at least, one common version of it exploits) a data-vs.-instructions vulnerability, or **NO** if the nature of the attack does *not* involve such a vulnerability. If uncertain, you might want to not circle either, as points will be deducted for circling the wrong answer.

—— **NO** TOCTTOU (time-of-check-to-time-of-use)

YES —— Buffer overflow

—— **NO** Integer overflow

YES —— SQL injection

—— **NO** TCP RST injection

—— **NO** DNS cache poisoning via Kaminsky's attack

YES —— Reflected XSS

—— **NO** CSRF

—— **NO** CAPTCHA "outsourcing" attacks

—— **NO** TCP SYN flooding

YES —— Exploiting use of `system()` or `popen()` instead of `execve()`

—— **NO** Clickjacking

Problem 2 ***Security Principles*** **(24 points)**

When shopping at Costco, after you've selected your purchases you take your cart full of goods to one of the registers. The check-out clerk scans your goods, totals what you owe, and upon receiving payment from you gives you an itemized receipt. However, you can't then simply exit the building with your goods. At the exit you're required to go by a staffmember who inspects your receipt. If the receipt looks okay (appears to match the number and types of items in your cart), the staffmember draws a line with a permanent marker down the receipt and hands it back to you. At this point, you can exit the building and take the goods to your car.

(a) (16 points) Identify two security principles illustrated by Costco's approach. For each, describe in a **single sentence** what aspect of Costco's approach reflects the principle.

> **Solution:** Two principles clearly apply. The use of two different staff members to handle payment/receipt processing illustrates *separation of responsibilities*. The fact that customers can only exit the building by passing through the second check illustrates *complete mediation*.
>
> That the user's payment is captured by one staff member and the receipt then double-checked by another plausibly illustrates *defense-in-depth*, though this is not quite as compelling a fit because it's not so clear that the register cashier provides "defense".
>
> A number of other principles can apply, too, if accompanied by thoughtful discussion.

(b) (8 points) Identify an attack that Costco seeks to prevent by having the staffmember draw the line down your receipt. Briefly describe how the attack works.

> **Solution:** One threat is a *replay attack* whereby a customer returns to the store and picks up another set of the same goods for which they've just paid. They then directly exit the building. If challenged to produce a receipt, they show the one from their previous trip.
>
> For an ecommerce attack of a similar flavor, see http://www.schneier.com/blog/archives/2011/05/vulnerabilities_2.html .

Problem 3 ***Modes of Encryption*** **(38 points)**

Consider the following encryption mode for applying AES-128 with a key K to a message M that consists of l 128-bit blocks, $M_1, \ldots, M_l$. The sender first picks a random 128-bit string, C_0, which is the first block of ciphertext. Then for $i > 0$, the i^{th} ciphertext block is given by $C_i = C_{i-1} \oplus \text{AES-128}_K(M_i)$. The ciphertext is the concatenation of these individual blocks: $C = C_0 \,||\, C_1 \,||\, C_2 \,||\, \cdots \,||\, C_l$.

(a) (6 points) What is the intent behind the random value C_0? (I.e., what is it meant to achieve.)

Solution: C_0 is an *Initialization Vector*. The intent behind using it is to ensure that if the same text is encrypted in two distinct messages, the ciphertexts will differ, so an eavesdropper can't infer the relationship between the messages.

(b) (20 points) Is this mode of encryption secure? If so, state what desirable properties it has that make it secure. If not, sketch a weakness.

Solution: It is not secure. Since the ciphertext is visible to an eavesdropper, the eavesdropper knows C_i for all values of i. This allows them to directly determine $\text{AES-128}_K(M_i)$ for all i due to the inverse nature of exclusive-or, which makes the scheme equivalent to ECB in terms of revealing whenever two message blocks contain the same text.

Another valid criticism is that because the scheme uses the Initialization Vector C_0 in a *reversible* manner, an attacker can deduce when two separate ciphertexts in fact encode the same text.

(c) (12 points) Suppose we replace the computation of C_i with $C_i = \text{AES-128}_K(C_{i-1} \oplus M_i)$. Does this make the mode of encryption more secure, less secure, or unchanged? Briefly explain your answer.

Solution: The mode is more secure. This alternate form is exactly the definition of CBC mode, which has been proven secure in the face of chosen plaintext attacks.

Problem 4 ***True/False*** **(35 points)**

For each of the following, circle **True** if the statement is correct, or **False** if it is not. If uncertain, you might want to not circle either, as points will be deducted for circling the wrong answer.

(a) (7 points) Cryptography:

——— **False** Computing a hash for a data item using a cryptographic hash function such as SHA-1 requires possession of the correct secret key.

True ——— A major problem with using one-time pads is distributing the keys.

——— **False** If an algorithm is discovered for quickly computing modular exponentiation, then today's public-key cryptography algorithms will become insecure.

True ——— Digital signatures use public-key cryptography to provide both integrity and authentication.

True ——— Caesar ciphers are vulnerable to known-plaintext attacks.

——— **False** Today's symmetric cryptography primitives aim to achieve resistance to known-plaintext attacks but not to chosen-plaintext attacks.

——— **False** When computing a MAC, it is vital to incorporate a randomized Initialization Vector that's unpredictable by an attacker.

(b) (7 points) TLS:

——— **False** If two web clients both retrieve the same URL from a given HTTPS server, then the bytes they transmit over the network to the server will be identical.

True ——— TLS uses a different key for encrypting traffic in the client-to-server direction versus encrypting traffic in the server-to-client direction.

——— **False** A TLS client confirms the validity of a certificate that a server sends it by verifying that the server signed the certificate.

True ——— TLS makes use of both asymmetric and symmetric cryptographic primitives.

——— **False** TLS provides protection against TCP RST injection attacks.

——— **False** A serious problem with TLS is that even after it becomes known that a server's private key has been stolen, there's no way for a client's browser to know that it should stop honoring the server's certificate.

True ——— Even if hardware is available to make all cryptography operations take essentially no time, fetching a given URL over HTTPS still will take a bit longer than it would using regular HTTP.

(c) (8 points) Viruses and Worms:

True —— How a virus spreads can be completely independent of the *payload* it executes on each system it infects.

True —— During their initial phase of propagation, well-designed worms can spread exponentially fast.

—— **False** A fundamental property of how viruses spread is that they generate random Internet addresses and then probe those to find new victims.

—— **False** One of the most promising approaches for defending against worm outbreaks is to release "counterworms" that spread by exploiting the same vulnerability, but upon infection de-install the original worm.

True —— A common approach for creating polymorphic viruses uses encryption technology.

True —— Viruses can spread to systems even if they have no Internet connectivity.

True —— Worms have spread that have compromised more than 10,000 Internet systems in less than an hour.

—— **False** If a worm uses random scanning of IP addresses to probe for new victims, then the time required for it to infect a target population increases linearly with the size of the population.

(d) (6 points) Detecting attacks:

—— **False** It is fundamentally harder to create a detector with a low rate of **false positives** than a low rate of **false negatives**.

True —— An advantage of **anomaly detection** over **signature-based detection** is the ability to potentially detect novel attacks.

True —— **Signature-based** techniques have the appealing property that it's easy to share the signatures between different parties.

—— **False** An advantage of *behavioral-based* techniques for detection is that they are well-suited for preventing attempted attacks from succeeding.

True —— Detectors have been demonstrated that never generate false positives, and likewise other detectors that never generate false negatives.

—— **False** To resist evasion, network-based detectors should analyze individual packets in a stateless fashion.

(e) (7 points) Privacy and Anonymity

True —— Web servers can instruct browsers to store a given cookie for many years.

—— **False** While there are some known theoretical attacks on it, if used carefully Firefox's *private browsing* mode will prevent web sites from discovering your name or IP address.

True —— Techniques exist that enable companies to track what text you cut-and-paste from certain web pages.

True —— California state law does not require companies to notify users of breaches of personal information *if* the data the attacker accessed was encrypted.

—— **False** "Web bugs" refer to controversial Javascript techniques that instruct web browsers to activate their computer's microphone and locally analyze the audio for certain keywords.

True —— Onion-routing schemes like the *Tor* anonymity network use a distinct cryptographic key for each hop that a given message takes through the network.

True —— When using a system like *Tor*, to ensure privacy you must route your DNS traffic through the system even if your computer always uses DNSSEC for its DNS lookups.

Problem 5 *eCommerce* **(24 points)**

Consider an ecommerce website that includes the notion of a "shopping cart." Customers visiting the site put items of interest in their shopping cart. After finishing their browsing and shopping, they click on `Checkout` to pay for the items. At that point, the customer logs into the site to enable the site to retrieve their payment information.

(a) (8 points) Suppose that the site implements the shopping cart by storing the associated items and prices in files on the server, with one file for each customer. The site identifies customers by their IP addresses.

This design is vulnerable to a DoS attack. Sketch it in a single sentence.

Solution: An attacker can continually add shopping cart items without end, exhausting the state available on the server for remembering the items.

(b) (16 points) Suppose that instead the site keeps a list of shopping cart items on the client side. Every time a user clicks on *add-to-cart*, the server sends all of the associated details (item name, price, quantity) in its reply, incorporating them into a hidden HTML form field. Through some Javascript magic, now when the user finally clicks on `Checkout`, all of the previously bought items embedded in the hidden form field are sent to the server. The server then joins them together into a list and presents the user with the corresponding total amount for payment.

1. Is this design vulnerable to the DoS attack you sketched above? Explain why or why not.

   Solution: This design is not vulnerable to the state-exhaustion attack because now the shopping cart state is all kept at the client. The attacker will exhaust their own memory, rather than the server's.

   It's valid to observe that the server might still be vulnerable to a DoS attack depending on what resources it requires for processing the list once the client proceeds to checkout.

2. Is this design secure from other attacks? If so, explain the basis for your claim. If not, describe an attack on it. (You can assume that the site is safe from web attacks such as CSRF, XSS and SQL injection, and uses HTTPS for the `Checkout` procedure.)

   Solution: It is not secure. As described, the server does not assure any *integrity* of the information stored on the client. An attacker can modify the information, in particular including the *price*.

Problem 6 ***Botnet C&C*** **(36 points)**

Consider the use of Twitter for botnet command-and-control. Assume a simplified version of Twitter that works as follows: (1) users register accounts, which requires solving a CAPTCHA; (2) once registered, users can post (many) short messages, termed *tweets*; (3) user A can *follow* user B so that A receives copies of B's tweets; (4) user B can tell when user A has decided to follow user B; (5) from the Twitter home page, anyone can view a small random sample (0.1%) of recent tweets.

(a) (12 points) Sketch how a botmaster could structure a botnet to make use of Twitter for C&C. Be clear in what actions the different parties (individual bots, botmaster) take. Assume that there is no worry of defensive countermeasures.

Solution: A simple approach is that the botmaster registers two Twitter accounts, A and B. (This requires only solving two CAPTCHAs, trivial to do by hand.) A is used to send commands, and B for receiving commands. The bot malware includes within it the credentials for the B account. New bots then access the B account to read the tweets sent by the A account, which encode the instructions to the bots.

An alternative approach would be to create a new account per bot. This requires some discussion of how to solve all the CAPTCHAs; for example, by purchasing solutions from a solving service available from the underground economy.

A third approach would be for the botmaster to register a single account and use it to generate 1000s of identical tweets for each command they want to send to their bots. Each bot visits the Twitter home page and examines the random sample of tweets there to look for instructions.

(b) (12 points) Briefly describe a method that Twitter could use to detect botnets using this C&C scheme.

Solution: For the first scheme, Twitter could look for access to the same account from many different IP addresses.

For the second scheme, Twitter could look for accounts whose followers all only follow that account.

For the third scheme, Twitter could filter out tweets from their random sample if the tweet is identical to a previous tweet.

(c) (6 points) How well will this detection method for Twitter method work?

Solution: The first scheme likely works well—it would be unusual for a legitimate account to be accessed form 1000s of different IP addresses.

The second scheme might run into trouble for certain types of users, say celebrities, who have many followers who only use Twitter to follow the celebrity.

The third scheme should work well—there's not much value in having an identical tweet show up in the random sample that's made public.

(d) (6 points) Briefly discuss a revised design that the botmaster could employ to resist this detection by Twitter.

Solution: For the first defense, the botmaster will need to shift to using one of the other two approaches given in part (a) above.

For the second defense, the botmaster could have the bots follow some other randomly selected users in order to look more normal.

For the third defense, the botmaster could add some minor variation to their repeated tweets so that Twitter doesn't view them as identical.

Problem 7 ***Surviving Hot Spots*** **(32 points)**

You return to Javalicious, the handy coffee shop nearby with free WiFi. You again settle in for an afternoon of web-surfing and tweeting. You know that the network sends all packets unencrypted, and you are not surprised to again see Prof. Evil seated at the table next to yours, using a laptop connected to the same WiFi network.

For your web connections, consider the basic security properties of *confidentiality*, *integrity*, and *availability*. For each of these, analyze three scenarios:

- **DNSSEC-only** means that for a given web site, your laptop looks up all of the domain names for your web session using DNSSEC (including NSEC3); your actual web traffic, however, uses HTTP.
- **HTTPS-only** means that for a given web site, your laptop looks up all of the domain names for your web session using ordinary DNS; your actual web traffic, however, uses HTTPS.
- **DNSSEC+HTTPS** means that both your domain name lookups use DNSSEC and your actual web traffic uses HTTPS.

In the following, circle **YES** if using only his laptop (no additional equipment) Prof. Evil can undermine the given property for your web connections, or **NO** if not. At the end of each section, supply a brief explanation for your answers.

(a) (8 points) **Confidentiality of your web connection content:**

Solution: YES for **DNSSEC-only**: the web traffic uses ordinary HTTP, which does not provide any confidentiality of its contents.

NO for **HTTPS-only** and **DNSSEC+HTTPS**, as HTTPS provides confidentiality. This assumes that you are prudent and do not continue with your web surfing if your browser alerts you to a certificate problem. Otherwise, Prof. Evil can launch a MITM attack and read the contents of your connections.

(b) (8 points) **Confidentiality of keeping private what sites you communicate with:**

Solution: YES for all three schemes. DNSSEC does not hide the names you look up nor the replies for those names. HTTPS does not mask your IP address nor that of the servers you visit.

(c) (8 points) **Integrity of your web connections:**

Solution: YES for **DNSSEC-only**, because HTTP does not provide integrity.

NO for **HTTPS-only** and **DNSSEC+HTTPS**, as HTTPS provides integrity. This again assumes that you are prudent and do not continue with your web surfing if your browser alerts you to a certificate problem.

(d) (8 points) **Availability of your web connections:**

Solution: YES for all three schemes. Prof. Evil can (for example) use TCP RST injection to terminate your connections before you are able to receive any web content. HTTPS protects *above* layer 4, but not layer 4 itself.

Problem 8 ***Information Leakage*** **(35 points)**

PlopIt! is a popular service that allows users to store files "in the cloud". For any file that a user wishes to make accessible to different Internet systems, the user uploads the file (via their browser) to *PlopIt!* and receives back a URL that provides direct access to the file. Each URL has the form `https://plopit.com/storage/`*user*`/`*hash*, where *user* is the name of the user who uploaded the file, and *hash* is the SHA-256 hash (64 hexadecimal digits) of the contents of the file. For example, such a URL might look like `https://plopit.com/storage/Alice/9b65...e7e6` . Users can then share these URLs with their friends or whomever they wish to allow to access the uploaded file. Users can also mark their uploads as "public," enabling anyone to view them via a browsing and search facility provided by *PlopIt!*.

(a) (12 points) Describe an attack on user security or privacy that this design enables. In your description, include mention of who might seek to launch the attack. Make as few assumptions about the attacker's capabilities as possible.

> **Solution:** Here's one of many possibilities. Suppose the RIAA suspects that Alice is distributing illegal copies of *Harry Potter and the Big Fat Residual Check*. They can compute S, the SHA-256 hash of the movie, and then try to fetch `https://plopit.com/storage/Alice/`S . If the fetch succeeds then they have proof that Alice did indeed upload the movie to *PlopIt!*, presumably for purposes of sharing it. This undermines Alice's privacy (putting aside the legality of her actions).
>
> Similarly, if there's any specific content you wonder that Alice may have stored in *PlopIt!*, this attack suffices to confirm its presence.

(b) (8 points) Describe a way that *PlopIt!* can defend against this attack. Your defense should require minimal changes on their part, and not disrupt their service model of enabling users to share files with friends.

> **Solution:** A simple solution for the attack above is for *PlopIt!* to introduce a *salt* when they compute hashes for uploaded files. Unless an attacker knows the value of the salt, they cannot compute the correct URL for a given item of content, and so won't succeed with this "confirmation" attack.

(c) (12 points) *PlopIt!* has become successful enough that they incur significant expenses for disk storage and network capacity. To reduce the volume of data they deal with, *PlopIt!* changes its upload mechanism to use compression, as follows. When a user wishes to upload a file, the uploader breaks the file into blocks. Before uploading a given block, the uploader sends a SHA-256 hash of the block's contents to see whether the server already has that block. If so, the browser avoids sending the block.

Given this compression (and assuming that *PlopIt!* uses the defense you developed

in the previous question), describe how two parties, Alice and Bob, can use *PlopIt!* to secretly communicate even though they have no means of directly sending information to one another. Assume that Alice and Bob had an opportunity in the past to agree on how their scheme will work—but they were *not* able to agree upon any specific *PlopIt!* URLs to use for communication.

For full credit, your scheme should enable Alice to transmit modest-sized messages (dozens to hundreds of bytes of data) to Bob without requiring a great deal of effort.

Your scheme: Solution: The key insight is that Bob can test whether a given block of data is already present on *PlopIt!* by attempting to upload that block and seeing whether the server responds with "that hash is already present, no need to upload it." This gives Bob a bit of information.

Alice and Bob can then agree upon a scheme of what blocks Bob should test for in order to recover different bits of Alice's message. For example, one scheme is that Alice will upload a file for which each block looks like: "`Bit` n `of Alice's message is a 1`" or "`Bit` n `of Alice's message is a 0`". She uploads one of these blocks, as appropriate, for each of $n = 1, 2, \ldots$ She can include all of the blocks in a single file (perhaps each with a lot of padding, depending on the blocksize used by *PlopIt!*), or as separate files. For each bit position, Bob then computes the corresponding hashes for the "... `is a 1`" and "... `is a 0`" cases, and tests which of those blocks is already present on the server.

A completely different approach is based on agreement of how to use *PlopIt!*'s search functionality. Alice and Bob can agree on a particular distinct search phrase; Alice uploads data for Bob that matches that phrase; and Bob then retrieves the data by issuing the search.

***Extra credit (6 points)*:** Describe how you can use your scheme to transmit GBs of data to Bob in a feasible amount of time.

Solution: Once Alice can transmit a message of a few dozen bytes to Bob using the first scheme outlined above, she can then use that channel to send Bob a *PlopIt!* URL that points to a file holding a very large amount of data.

For the second scheme above, this is even easier: the search functionality can pull up a very large file that Alice has stored in *PlopIt!*.

(d) (3 points) Suppose that a *warden* monitors Bob and prevents Bob from engaging in *any* use of the *PlopIt!* site. The warden allows Alice to send *PlopIt!* URLs to Bob, however, because the warden likes to taunt Bob with the warden's censorship of the site. The warden does not allow Alice to send *anything* else to Bob.

Describe how Alice and Bob can have agreed upon a scheme in advance that allows Alice to send an n-byte message to Bob by sending Bob a total of n *PlopIt!* URLs. Assume that the warden validates each URL to make sure that it does indeed point to a file stored in *PlopIt!*.

Solution: Here the key insight is that Alice and Bob can agree to treat part of the hash present in the URL as data Alice wants to send to Bob. For example, they can agree that every time Alice sends a URL to Bob, the bottom two hexadecimal digits of the hash will encode the next byte of Alice's message.

Alice now needs to put together a set of URLs that have hashes for which the bottom hexadecimal digits match the different bytes she wishes to transmit. She can do this by simply uploading a series of files with different contents, observing the hashes that *PlopIt!* assigns to them (this works even if *PlopIt!* adds salting to its hashes). It won't take more than a few hundred such file creations for Alice to have a complete set of URLs that amongst them have all possible pairs of final hexadecimal digits. She then for each byte of her message selects the URL that encodes that particular byte and sends it to Bob. The warden will verify that the URL is indeed a valid *PlopIt!* URL and allow Bob to receive it.

Problem 9 ***Cryptographic Properties*** **(40 points)**

For this problem, assume that Alice wants to send a *single* message M to Bob. To do so, Alice and Bob can potentially use a number of different approaches and cryptographic technologies, which we will describe using the following terminology:

M	Plaintext for a single message
s_k	Symmetric cryptography key
AES_{s_k}	Symmetric-key encryption using AES-256 in CBC mode, with the key s_k
PRNG_{s_k}	Bit-stream from a cryptographically strong pseudo-random number generator, seeded with s_k
SHA_{256}	SHA-256 hash function
AES-EMAC_{s_k}	Keyed MAC function presented in lecture, using the key s_k
K_A	Alice's public key
K_A^{-1}	Alice's corresponding private key
K_B	Bob's public key
K_B^{-1}	Bob's corresponding private key
E_K	Public-key encryption using RSA with the public key K
$\text{Sign}_{K^{-1}}$	Public-key decryption using RSA with the private half of K. For its use as a building block for a digital signature, you do not need to worry about padding issues.

You can assume that the public keys have been securely distributed, so Alice and Bob know their correct values.

Consider the following properties that Alice and Bob might desire their communication to have: *Confidentiality*, *Integrity*, and *Non-Repudiation*. For each of the following possible communication approaches, **circle** which of these properties will securely hold in the presence of Mallory, a MITM attacker. Circle **None** if *none* of the properties hold. If an approach fails entirely (will not result in Bob being able to read a given message M), circle **Broken** (and do not bother with the others).

(a) (5 points) Alice generates a new symmetric key s_k and sends to Bob: $E_{K_A}(s_k)$, $E_{K_B}(s_k)$, $M \oplus \text{PRNG}_{s_k}$

Solution:

This scheme only provides *Confidentiality*. While Bob cannot recover s_k from $E_{K_A}(s_k)$ (because Bob lacks Alice's private key), he can do so from $E_{K_B}(s_k)$. The encryption using $M \oplus \text{PRNG}_{s_k}$ is the same as a stream cipher, which will indeed provide confidentiality. However, without a separate MAC, the communication lacks integrity, and because Alice does not sign her message, it also lacks non-repudiation.

Note that confidentiality for *multiple* messages is undermined by the lack of use of an Initialization Vector. However, the problem framing specifically discusses

Alice sending a *single* message.

(b) (5 points) Alice generates a new symmetric key s_k and sends to Bob: $E_{K_A}(s_k)$, $E_{K_B}(s_k)$, $\text{AES}_{s_k}(M)$

Solution: The only difference between this scheme and the previous one is the use of AES rather than the stream cipher. By itself, AES does not provide integrity, so this scheme likewise only provides *Confidentiality*.

(c) (5 points) Alice sends to Bob: $E_{K_A}(M)$, $\text{Sign}_{K_A^{-1}}(\text{SHA}_{256}(M))$

Solution: Broken: to decrypt with this scheme, Bob needs to possess Alice's private key.

(d) (5 points) Alice sends to Bob: $E_{K_B}(M)$, $\text{Sign}_{K_B^{-1}}(\text{SHA}_{256}(M))$

Solution: Broken: this scheme requires Alice to possess Bob's private key for the signing operation.

(e) (5 points) Alice sends to Bob: $E_{K_A}(M)$, $\text{Sign}_{K_B^{-1}}(\text{SHA}_{256}(M))$

Solution: Broken: to decrypt with this scheme, Bob needs to possess Alice's private key. Alice also needs to possess Bob's private key for the signing operation.

(f) (5 points) Alice sends to Bob: $E_{K_B}(M)$, $\text{Sign}_{K_A^{-1}}(\text{SHA}_{256}(M))$

Solution: This solution provide all of *Confidentiality* (via the encryption using Bob's public key), *Integrity* (via the digital signature over the hash of the message), and *Non-Repudiation* (via Alice using her private key for the digital signature).

(g) (5 points) Alice and Bob privately exchange a symmetric key s_k in advance. Alice later uses this key to send to Bob: $\text{AES}_{s_k}(M)$, $\text{AES-EMAC}_{s_k}(\text{SHA}_{256}(M))$

Solution: This scheme provides *Confidentiality* (via the use of AES) and *Integrity* (via use of the keyed MAC function). It does not provide non-repudiation because the integrity/authentication component does not demonstrate possession of Alice's private key.

That said, a legitimate criticism of this approach is the reuse of the same key for encryption and the MAC computation, which may make it easier to break the secret key.

(h) (5 points) Alice generates a new symmetric key s_k and sends to Bob: $E_{K_A}(s_k)$, $E_{K_B}(s_k)$, $\text{Sign}_{K_A^{-1}}(s_k)$, $\text{AES}_{s_k}(M)$

Solution: The crucial insight for this problem is that Alice's signature over s_k allows Mallory to recover s_k simply by computing $E_{K_A}(\text{Sign}_{K_A^{-1}}(s_k))$, which Mallory can easily do since K_A is well-known. Given possession of s_k, all of the properties fail to hold: Mallory can read the message and can alter it, so there is no confidentiality and no integrity. There is no non-repudiation, either; all that the signature can demonstrate is that Alice signed s_k, but not that she signed M.

However, Bob can still recover M. Therefore this scheme is marked **None**, rather than **Broken**.