Final Solutions

Question	Parts	Points
1: True or False	9	18
2: Collision Resistance	4	14
3: Signatures	4	22
4: Messaging security	1	20
5: Open problems in encryption	3	14
6: Isolation	3	18
7: Guest lecture: Network security	2	6
8: Software trust	1	20
9: Fuzzing	1	16
10: Lab 3	1	16
11: Lab 4	1	16
Total:		180

Name: ______________________________

- This is an open book exam: you can use your notes or any material released by us this term. You cannot use the internet.
- Any form of collaboration is *strictly* forbidden.
- If you find a question ambiguous, be sure to write down any assumptions you make.

This exam is printed double-sided!

Problem 1. [18 points] **True or False** (9 parts)
Please write **T** or **F** for the following. *No justification is needed (nor will be considered).*

(a) [2 points] Given a secure pseudorandom function, it is possible to build a secure MAC.

Solution: True.

(b) [2 points] Let $H\colon \{0,1\}^n \to \{0,1\}^n$ be a one-way function. The function H must have collisions.

Solution: False.

(c) [2 points] The SSH wire protocol completely hides the length of the message that the client and server exchange.

Solution: False.

(d) [2 points] In Schnorr's signature scheme, an attacker can use the extractor to extract the secret key from the public key.

Solution: False.

(e) [2 points] Secure boot protects an iPhone from vulnerabilities in the boot ROM.

Solution: False.

(f) [2 points] The "rowhammer" effect in DRAM can cause memory corruption but cannot lead to security issues.

Solution: False.

(g) [2 points] Diffie-Hellman key exchange over Z_p^* with key size of 1024 bits is strong enough for most use cases over the Internet.

Solution: False.

(h) [2 points] In practice, when signing messages with RSA, it is critical to check if your signature is valid, before returning it.

Solution: True.

(i) [2 points] ECDSA signatures (using 128-bit security) are shorter than RSA signatures (using 128-bit security).

Solution: True.

Problem 2. [14 points] **Collision Resistance** (4 parts)
Let $H\colon \{0,1\}^{2n} \to \{0,1\}^n$ be a hash function, which you can model as a truly random function.

(a) [2 points] Give one example of a collision-resistant hash function used in practice.

Solution: SHA-256

(b) [4 points] What is the smallest possible choice of the hash-output length n that ensures that a collision-finding attack will take at least, roughly, 2^T time? (Your answer should be correct up to logarithmic factors in T.)

Solution: $n \geq 2T$

Contined on the next page…

(c) [4 points] Define a k-*almost-collision* to be a pair of distinct strings $m_0, m_1 \in \{0,1\}^{2n}$ such that the XOR of their hashes $H(m_0) \oplus H(m_1) \in \{0,1\}^n$ begins with k zeros; the remaining $n-k$ bits can be anything.

How much time as a function of n and k, does it take to find a k-almost collision in H? (Your answer should be correct up to polynomial factors in n and k.)

Solution: $2^{k/2}$

(d) [4 points] How much time, as a function of n, does it take to find distinct strings $x_0, x_1 \in \{0,1\}^n$ such that $H(x_0||x_1) = \underbrace{000\cdots000}_{n \text{ times}}$?

(Your answer should be correct up to polynomial factors in n.)

Solution: 2^n

Problem 3. [22 points] **Signatures** (4 parts)

(a) [3 points] Explain in one sentence what a one-time-secure signature scheme is.

Solution: Secure when used to sign at most one message (but can verify many times).

(b) [3 points] Explain in one sentence why Lamport signatures are only one-time secure.

Solution: An adversary that learns signatures on two distinct messages that differ in many bits (e.g., the all-zero and all-ones messages) can choose any combination of the bits in which the messages differ, generating valid signatures for messages that were not signed.

(c) [8 points] Your friend proposes reducing the size of Lamport signatures by having the signer drop the most-significant bit of the signature. (So, when using a hash function over k-bit inputs, and signing an n-bit message, the original $n \cdot k = \ell$-bit signature becomes an $(\ell - 1)$-bit signature.) Explain how a verifier can verify this compressed signature.

Solution: Check if adding back either bit leads to a valid signature.

(d) [8 points] Explain how to extend your friend's scheme to compress a Lamport signature on an n-bit message by n bits. (That is, an ℓ-bit signature becomes an $(\ell - n)$-bit signature.) The verifier should run in time *polynomial* in n.

Solution: Drop one bit from each hash pre-image, so that each of the n omitted bits can be independently recovered in two tries, for $2n$ tries total to "de-compress".

Problem 4. [20 points] **Messaging security** (1 part)
Ben Bitdiddle is building an encrypted messaging system, but after trying to understand the ratchet protocol, he thinks it's too complicated, and he proposes a simpler alternative. Assume that users, such as Alice and Bob, have secret keys (which we will denote a and b) and know each other's public keys (g^a and g^b). In Ben's protocol, each time that Alice wants to send a message to Bob, Alice computes a shared secret between her and Bob, as $s = (g^b)^a$, and encrypts her message using authenticated encryption, with the key computed as $H(s)$, where H is a cryptographic hash function. When Bob receives this message from Alice, Bob computes the shared secret s as $(g^a)^b$, and then uses the authenticated encryption scheme, with key $H(s)$, to authenticate and decrypt Alice's message.

Describe at least two weaknesses in Ben's scheme as compared to the ratchet construction.

Solution: This construction uses the same key to encrypt messages from Alice to Bob and Bob to Alice. The adversary could send Alice's message back to Alice, and it will appear to be sent by Bob.

This construction does not offer forward secrecy.

This construction does not offer post-compromise security.

Problem 5. [14 points] **Open problems in encryption** (3 parts)

(a) [6 points] Say that a client and server communicate over a network. An attacker controls a network router between the client and server. For each of the following, answer TRUE or FALSE:

- If the parties communicate over HTTP, the attacker can tamper with the bytes exchanged between the two parties without detection.

  Solution: True.

- If the parties communicate over SSH, the attacker can tamper with the bytes exchanged between the two parties without detection.

  Solution: False.

- If the parties communicate over TLS, the attacker can prevent the client from communicating with the server.

  Solution: True.

(b) [4 points] A client communicates with a server over TLS. List two types of information about this communication flow that leak to an eavesdropper on the network between the client and server.

Solution: Timing, data size.

(c) [4 points] For each of the two types of information you listed in part (b), state whether the client can hide this information by routing its traffic through Tor, and explain why or why not.

Solution: No, no.

Problem 6. [18 points] **Isolation** (3 parts)
Consider the following scenarios in which a single computer runs two pieces of code, one from Alice and one from Bob. For each scenario, state whether the system provides non-interference, non-leakage, or none of the above, and explain your answer. Assume there are no bugs in the implementation of the system.

(a) [6 points] Alice and Bob's code is run as different virtual machines (VMs) on the same physical CPU. The VMs have access to the network.

Solution: Non-leakage. Non-interference would be hard to achieve due to side channels such as timing.

(b) [6 points] Alice and Bob's code is run as two different modules in a single Python application.

Solution: No isolation.

(c) [6 points] Alice and Bob's code is run in two WebAssembly sandboxes, using the pure Python implementation of WebAssembly from lab 4, without access to the file system and without the ability to allocate more memory at runtime.

Solution: Non-interference; no side channels.

Problem 7. [6 points] **Guest lecture: Network security** (2 parts)

(a) [3 points] Name the computer-security concern that Michael Duff (Harvard's Chief Information Security Officer) spends most of his time worrying about.

Solution: Phishing

(b) [3 points] Why is damage to Harvard's reputation such a major concern for Michael Duff?

Solution: Might affect fund-raising or recruiting students.

Problem 8. [20 points] **Software trust** (1 part)
Ben Bitdiddle is working on a game console with secure boot, but wants to ensure that previously-signed old kernel images cannot be booted (because they might contain known security vulnerabilities). He proposes the following design, in pseudo-code:

```
global boot_pk = ...

def boot_loader():
  helper = read_from_disk("/helper")
  kernel = read_from_disk("/kernel")
  r = generate_random_challenge()
  sig = helper(r, kernel)
  if not verify(boot_pk, r + kernel, sig):
    raise Exception("invalid sig")
  kernel()
```

The boot loader generates a challenge and runs a *helper* kernel whose job is to communicate with a server, over the network, to get a signature on the concatenation of the random challenge and the kernel image. The server will only sign this challenge concatenated with the latest kernel image. When the helper obtains the signature from the server, it returns the signature back to the boot loader. The boot loader, in turn, checks the signature and boots the kernel if the signature validates. The pseudo-code above informally treats the `helper` and `kernel` variables both as strings (e.g., when validating the signature on the kernel image) as well as functions (e.g., when running the machine code corresponding to the helper or kernel).

Can an adversary that can tamper with the contents of the disk boot an old vulnerable OS in Ben's design? Explain why or why not.

Solution: Ben's scheme is not secure: adversary could just run arbitrary code in the helper kernel. Or pass an old challenge value r back to `helper_done`, together with a signature for that old challenge value.

Problem 9. [16 points] **Fuzzing** (1 part)
Ben Bitdiddle is writing an HTML layout engine, whose job is to figure out how to render a particular web page on the screen. Ben wants to use a fuzzer to find bugs that cause his layout engine to crash, but when he tries using a fuzzer to generate random input files, he doesn't find any bugs at all. Explain what's going wrong and how Ben can use the fuzzer more effectively.

Solution: Random files aren't valid HTML, and random changes to HTML also doesn't produce particularly significant layout differences. Perhaps it would be more fruitful to build a pseudo-random HTML generator, which picks from a set of HTML elements, layout attributes, etc.

Problem 10. [16 points] **Lab 3** (1 part)
Ben Bitdiddle implemented the attack for lab 3 problem 3 part a (decrypting compressed data known to be a JSON struct containing 3 capital city names), and this gave him an idea for a defense against this attack. Ben modifies SSH so that compressed data is rounded up to the next kilobyte before being sent. This way, if the compressed data is smaller than 1 KByte, as in his attack, the protocol always sends 1 KByte. If the data is larger than 1 KByte but less than 2 KBytes, the server will send 2 KBytes, etc.

Describe how an adversary can still implement an attack against Ben's modified SSH protocol.

Solution: Prepend a long random prefix of a little less than 1 KByte in length, so as to get the compressed data on the threshold of overflowing into the next kilobyte. Perform the original attack, and for each probe, see how much extra data needs to be prepended at the beginning to observe an overflow from 1 KByte to 2 KByte.

Problem 11. [16 points] **Lab 4** (1 part)
Ben Bitdiddle is implementing lab 4 problem 2 part 3 (sandboxing the SHA-256 code using WebAssembly). His implementation is, roughly:

```
def sha256(v):
    ...
    inptr = runtime.exec('malloc', [len(v)])
    outptr = runtime.exec('malloc', [32])
    runtime.store.memory_list[0].data[inptr] = v
    runtime.exec('SHA256', [inptr, len(v), outptr])
    return bytes(outptr)
```

Explain what two issues Ben needs to fix in order to make his code work.

Solution: Ben does not correctly copy the input data `v` into the sandbox memory, and does not copy the result back at all.