Final Solutions

Question	Parts	Points
1: True or False	10	20
2: User authentication	3	15
3: Symmetric Encryption	2	10
4: Public-Key Encryption	1	10
5: Compress then encrypt	3	15
6: Certificate revocation	2	10
7: Access control	1	5
8: Secure boot	1	5
9: LPN Error Correction	1	5
10: Isolation	2	10
11: Fuzzing	2	10
12: Symbolic execution	2	15
13: Sandbox Sharing	3	15
14: Differential Privacy	1	5
15: Timing Side Channel	4	20
16: Course Survey	4	10
Total:		180

Name: _____

Problem 1. [20 points] **True or False** (10 parts)
Please circle **T** or **F** for the following. *No justification is needed (nor will be considered).*

(a) [2 points] One way to construct a CPA-secure encryption scheme is with the "hash-and-encrypt" paradigm.

Solution: False.

(b) [2 points] The RSA signature scheme is post-quantum secure.

Solution: False.

(c) [2 points] One way to construct a secure signature scheme for arbitrary length messages is with the "hash-and-sign" paradigm.

Solution: True.

(d) [2 points] Transport-layer security is the protocol that underlies HTTPS.

Solution: True.

(e) [2 points] A CCA-secure encryption scheme must also be CPA-secure.

Solution: True.

(f) [2 points] If a client and server communicate using transport layer security (TLS), an eavesdropper in the middle of the network can learn the IP addresses of the client and server.

Solution: True.

(g) [2 points] A CCA-secure encryption scheme hides the length of the plaintext.

Solution: False.

(h) [2 points] It is impossible to update the lowest level Boot Read-Only Memory (ROM) in an iOS device.

Solution: True.

(i) [2 points] If you live in Belgium and connect to the Internet via a U.S.-based VPN provider, your Belgian ISP learns no information about your browsing behavior.

Solution: False.

(j) [2 points] You connect to a Google server via an encrypted TLS connection. Your ISP can distinguish between the cases in which (a) you watch a 30-minute episode of Stephen Colbert on YouTube and (b) you watch a 30-second cat video.

Solution: True.

Problem 2. [15 points] **User authentication** (3 parts)
An adversary has compromised the password databases from 128 (2^7) different servers, each with 1M (2^{20}) users. The databases all use hashing and salting for passwords, with 24-bit salts (chosen at random). Suppose users choose passwords with 10 bits of entropy (i.e., you can think of them as being random 10-bit values). Assume the adversary has not done any pre-computation before embarking on their attack. Assume that the password DB compromised by the adversary contains the salts. For all answers in this question, approximations within a factor of 2 are OK.

(a) [5 points] How many hashes will the adversary need to compute in order to guarantee guessing the password of a specific user on a specific server?

Solution: $2^{10} = 1024$.

(b) [5 points] How many hashes will the adversary need to compute in order to guarantee guessing the passwords of all users on a specific server?

Solution: $2^{10} \times 2^{20} = 2^{30}$; maybe a bit less for a few salt collisions at random, but within a factor of two. The number of colliding salts should be within $2^{20-24} \times 2^{20} = 2^{16} << 2^{20}$.

(c) [5 points] How many hashes will the adversary need to compute in order to guarantee guessing the passwords of all users on all servers?

Solution: Brute-force each password separately: $2^7 \times 2^{20} \times 2^{10} = 2^{37}$.
Compute complete table of possible salted passwords: $2^{10} \times 2^{24} = 2^{34}$. This is the way to go.

Problem 3. [10 points] **Symmetric Encryption** (2 parts)

(a) [5 points] Construct a CPA secure symmetric encryption scheme for messages in $\{0,1\}^{256}$ given a CPA secure symmetric encryption scheme for messages in $\{0,1\}^{128}$.

Solution: To encrypt a message $M \in \{0,1\}^{256}$ simply parse the message into $M = M_1||M_2$, where $M_1, M_2 \in \{0,1\}^{128}$ and outputs $Enc(sk, M_1), Enc(sk, M_2)$, where Enc corresponds to the underlying scheme for messages in $\{0,1\}^{128}$.

(b) [5 points] Show how to upgrade the security to CCA security given a MAC scheme for messages of arbitrary length.

Solution: To upgrade this scheme to CCA secure simply add a MAC to the outputted ciphertext.

Problem 4. [10 points] **Public-Key Encryption** (1 part)
Consider a 2-message key exchange protocol, where party 1 chooses randomness r_1 and sends a message m_1, which is a function of r_1 (i.e., $m_1 = m_1(r_1)$), party 2 chooses randomness r_2 and replies with a message m_2, which is a function of m_1 and r_2 (i.e., $m_2 = m_2(r_2, m_1)$), and the shared secret s can be efficiently computed from (m_1, r_2) or (m_2, r_1).

Construct a CPA secure public-key encryption scheme from the key exchange protocol above, where the *weak* security guarantee of the key-exchange protocol is that given (m_1, m_2) generated as above, it is hard to compute the shared secret s (but s is not necessarily indistinguishable from uniform given (m_1, m_2)).

You can assume (for simplicity) that the secret key s is a binary string of length k (for some k), and that there is a secure hash function $H : \{0,1\}^k \to \{0,1\}^{k'}$ (modelled as a Random Oracle), and construct an encryption scheme for messages in $\{0,1\}^{k'}$.

Solution: The KeyGen algorithm is the same as above: It samples randomness r_1, computes $m_1 = m_1(r_1)$, and outputs $(sk, pk) = (r_1, m_1)$. The encryption algorithm given $pk = m_1$ and a message m to encrypt, chooses randomness r_2, computes s from (m_1, m_2, r_2) and outputs $(m_2, m \oplus H(s))$ as the encryption of m. The decryption algorithm given $sk = r_1$ and a ciphertext $(m_2, m \oplus H(s))$, uses m_2, r_1 to compute s. Then it computes $H(s)$ and uses this to unmask $m \oplus H(s)$, and thus obtain the decrypted message m.

Problem 5. [15 points] **Compress then encrypt** (3 parts)
You browse to `evil.com` while using an evil WiFi network, so the attacker can run JavaScript in your browser *and* observe your network traffic.

While you are on this evil website, the attacker's JavaScript causes your browser to load a sequence of URLs from Google. That is, the attacker can cause your browser to send the following type of `GET` request to Google.

```
GET /ATTACKER_CHOOSES_THIS_URL HTTP/1.1
Host: google.com
Accept: */*
Cookie: SECRET_VALUE_HERE
```

The Google cookie value is a secret 32-byte hexadecimal string—e.g., `1FAB8AEEC9`$\cdots$`9809D9BC02`. (Each hexadecimal character in ASCII takes one byte to represent.)

The `GET` request travels to Google over an encrypted TLS connection. Recall that TLS encryption increases the length of the plaintext by a fixed amount.

Say that your browser *compresses* the `GET` request string before encrypting it. The compression algorithm has the property that:

- If there is no repeated 4-byte string in the `GET` request, the output is L bytes long.
- If there are r repeated 4-byte strings in the `GET` request, the output is $L - r$ bytes long.

(a) [5 points] Show that if the attacker can guess the first 4 bytes of your Google cookie value, it can confirm that its guess was correct.

> **Solution:** If your guess for the first 4 bytes is `ABCD`, make a `GET` request for URL `/ABCD`. Then look at the length of your encrypted packet. If the guess was correct, the packet will be $c + L - 1$ bytes long, for a known constant c. If not, the packet will be $c + L$ bytes long.

(b) [5 points] Show that if the attacker can cause your browser to load 2^{16} URLs, it can recover the first 4 bytes of your Google cookie.

Solution: Run the attack of part (a) on all possible 4-byte hex strings.

(c) [5 points] Show how the attacker can recover your entire Google cookie by loading $\ll 2^{20}$ URLs. You may assume that the cookie itself contains no repeated 4-byte strings.

Solution: Use part (b) to get the first 4 bytes. Then recover the next 4 bytes by using part (b) again and again until you have the entire cookie. This takes $8 \cdots 2^{16}$ fetches at most.

Problem 6. [10 points] **Certificate revocation** (2 parts)
This problem deals with certificate revocation in the public-key infrastructure.

(a) [5 points] The *online certificate status protocol* (OCSP) works as follows: when your browser gets a public-key certificate from a server, the certificate contains the URL of an OCSP server, typically run by the certificate authority who issued the certificate. The client then asks the OCSP server "Is this certificate `<hash of cert>` still valid?" The OCSP server responds with a signed message indicating YES or NO.

Unfortunately, the OCSP server learns which websites the client is visiting. A more recent technology, called *OCSP stapling* has the web server (e.g., `nytimes.com`) fetch a signed OCSP response from its certificate authority indicating that its certificate is still valid. This eliminates the privacy concern from the original scheme, but requires the web server to periodically fetch a new signed attestation from the certificate authority. The OCSP response includes a timestamp and clients will reject the response if it is too old.

Yet another concern with OCSP that it adds latency to the client's web browsing experience, since the client must contact the OCSP server before it completes its connection to the web server.

To reduce client latency, one option would be to first download the webpage from the web server (e.g., `nytimes.com`) and then use OCSP to check the validity of the certificate. If the certificate is valid, the browser would tear down the browser window and display an error.

What security problems could arise from using OCSP like this?

Solution: If the certificate was revoked because the private key was leaked, the webpage could potentially contain malware from a network attacker. By the time the OCSP check completes, the client's browser may have already executed the malware (e.g., stealing the client's cookies, etc.).

(b) [5 points] An alternative to OCSP is certificate revocation lists (CRLs). When using CRLs, the certificate authority includes in each certificate the URL of a CRL file. The client can fetch this signed CRL file, which contains a list of all certificates that the certificate authority has revoked. In this way, the client can check whether a particular certificate has been revoked.

Explain how a malicious certificate authority could abuse this process to learn which clients are visiting which websites.

Solution: Put a distinct CRL URL in each certificate. Then, by looking at which clients are visiting which CRL URLs, you can learn which client is looking at which webpage.

Problem 7. [5 points] **Access control** (1 part)
Ben Bitdiddle is designing a data storage system, and thinks capabilities are great because they allow for easy delegation. He uses capabilities as the basic access control primitive in his storage server, in lieu of traditional access control lists: clients must have a capability in order to fetch an object from his data store.

What problems might Ben run into by using only capabilities?

Solution: Might be hard to track down who has access to any particular piece of data.

Might be hard to revoke capabilities once they leak out.

Problem 8. [5 points] **Secure boot** (1 part)
Alyssa P. Hacker is building a game console and wants to make sure that the game console hardware can only be used to run authentic games approved by her company. Alyssa wants to achieve this using secure boot, but wants to avoid checking many signatures during boot, and wants to avoid having to hard-code a public key at each step of the boot process. Alyssa's idea is to have each boot step store a hash of the next step in a designated memory location, and then the final boot loader will be responsible for checking a signature on all of these hashes together before jumping to the OS kernel.

Specifically, the boot ROM loads the boot loader from disk, computes a hash of the boot loader, h_{boot} and stores it in a designated memory location m_{boot}, before jumping to the boot loader. The boot loader then loads the OS kernel from disk, computes a hash h_{os} and stores it in designated memory location m_{os}. Then, before jumping to the OS kernel, the boot loader loads the signature from disk and checks that it's a valid signature, under the public key embedded in the boot loader, of $h_{boot}||h_{os}$. If the signature matches, the boot loader jumps to the OS kernel and proceeds booting. Otherwise, the boot stops.

Is Alyssa's scheme secure? Explain why or why not.

Solution: Not secure because the second step (boot loader) can be modified on-disk with a version that does not do any signature checks at all.

Problem 9. [5 points] **LPN Error Correction** (1 part)
Recall the LPN based error correction scheme (Lecture 17). n refers to the number of bits in the secret key s, m is the number of ring oscillator pairs and the number of bits in the helper data b and noise vector e.

The **Gen** step determines the b vector and exposes it. The **Rep** step chooses the n most stable bits (choosing the n out of m counter values that are the largest absolute values regardless of sign), and uses the corresponding n equations to solve for s.

Ben Bitdiddle (remember him from 6.004?) decides that he can retain his five customers by marketing his new scheme as providing **independent** secrets $s^{(k)}$ for each of his customers, k varying from 1 to 5. That is, he will use the same ring oscillator circuit array to "encode" different $s^{(k)}$ values into m-bit $b^{(k)}$ vectors in the **Gen** step. The **Rep** step is unchanged, given a $b^{(k)}$ vector the circuit regenerates $s^{(k)}$. Ben claims that his scheme is more general than the original scheme and has equivalent security. Is Ben correct? Provide a brief explanation.

Solution: No. The LPN problem assumes that the e_i values are hidden noise, that is the e_i values are independently generated. Ben is *reusing* e_i values across his k customers so the problem the adversary faces is not strictly LPN. Note that the $s^{(k)}$ values **are** independent of each other, it is the repeated noise that results in weaker security.

Problem 10. [10 points] **Isolation** (2 parts)

(a) [5 points] Ben Bitdiddle is using virtual machines for isolation. He notices that the cost of saving and restoring registers when switching between virtual machines is a significant source of overhead for his workload. Furthermore, he notices that a large part of that cost is due to saving large floating-point registers, even if the virtual machine didn't use any floating-point code.

Ben implements an optimization: instead of saving and restoring floating-point registers when switching between virtual machines, his virtual machine monitor lets the floating-point registers remain in-place. However, he configures the processor to trap into the virtual machine monitor if the virtual machine runs any instructions that access a floating-point register. At that point, his virtual machine monitor would finally save and restore the floating-point registers as it would have originally done, and enables the use of floating-point instructions.

Is this optimization secure with respect to the virtual machines providing isolation? You may assume non-speculative hardware processors. Explain why or why not.

Solution: Two different answers were accepted, one for yes and one for no.

On the one hand, there is no way for one virtual machine to observe or modify the contents of the floating point register state of another virtual machine, because either the floating-point instructions are disabled, or the correct FP register state is present.

However, the very fact that one virtual machine used floating-point instructions may be leaked to another VM via a timing side channel, as the first FP instruction will take longer in this case.

(b) [5 points] Ben allows virtual machines to dynamically manage their memory allocations. In particular, he adds special calls from the virtual machine into the virtual machine monitor that allow the VM to either allocate more pages of memory, or to give some pages of memory back to the shared global set of free pages. When allocating or freeing memory, the virtual machine monitor ensures that page tables are properly updated, so that at most one virtual machine has a page of memory mapped in its page tables at a time.

Does this design provide strong non-interference between virtual machines? Explain why or why not.

Solution: No: virtual machines will be able to observe when the system runs out of free pages.

Maybe also no if the virtual machine monitor forgets to clear the contents of pages when re-allocating them from one VM to another.

Problem 11. [10 points] **Fuzzing** (2 parts)
Consider the following C function being tested by a fuzzer that supplies random inputs in the 32-byte `input` array:

```
unsigned char input[32];

void f() {
  if (input[0] != input[1])
    return;

  if (input[2] < 128)
    return;

  if (input[3] > 32)
    crash();
}
```

(a) [5 points] How many inputs, on average, will a purely random (not coverage-guided) fuzzer need to trigger the call to `crash()`?

Solution: The probability of any random input triggering a crash is $\frac{1}{256} \times \frac{128}{256} \times \frac{224}{256} = \frac{224}{131072} = \frac{7}{4096} \approx 0.17\%$. So, will need about $\frac{4096}{7}$ inputs.

(b) [5 points] How many inputs, on average, will a coverage-guided fuzzer need to trigger the call to `crash()`?

Solution: On average 128 inputs to get past the first if statement, then 1-2 inputs to get past the second if statement, then 1 more input to get past the third if statement, for a total of 130-131.

Problem 12. [15 points] **Symbolic execution** (2 parts)
Consider the following C function being executed in a symbolic execution system with an arbitrary symbolic argument x:

```
void f(unsigned int x) {
  if (x % 2 == 0) {
    x = x + 1;
  }

  if (x > 4096) {
    return;
  }

  if (x < 16) {
    crash();
  }
}
```

(a) [10 points] Which of the following queries will a symbolic execution issue to its SAT solver? Check all that apply.

1. $((x \mod 2) == 0)$
2. $\neg((x \mod 2) == 0)$
3. $(x > 4096)$
4. $\neg(x + 1 > 4096)$
5. $(x + 1 > 4096) \wedge ((x \mod 2) == 0)$
6. $(x + 1 > 4096) \wedge \neg((x \mod 2) == 0)$
7. $(x < 16) \wedge (x > 4096) \wedge ((x \mod 2) == 0)$
8. $(x < 16) \wedge \neg(x > 4096) \wedge \neg((x \mod 2) == 0)$
9. $\neg(x + 1 < 16)$
10. $\neg(x < 16) \wedge (x > 4096) \wedge \neg((x \mod 2) == 0)$

Solution: 1, 2, 5, 8

(b) [5 points] How many execution paths will the symbolic execution engine explore in executing `f(x)`?

Solution: $2\times$ due to the first if statement. After the first if statement, three paths: either the second if statement returns, or the third if statement crashes, or the function returns after the third if statement. So, 6 paths total.

Problem 13. [15 points] **Sandbox Sharing** (3 parts)
Consider the following system where public keys identify users. Users own hardware authentication tokens running a Python host program. In this process is embedded a WebAssembly instance which owns the private key and signs messages on the user's behalf. The instance's source code is `auth.c` (implementation omitted):

```
static void uint8_t private_key[32];
void key_gen(uint8_t public_key[32]);
void hash(const uint8_t *message, uint16_t message_len,
          uint8_t hash_out[32]);
void sign(const uint8_t message_fixed[32], uint8_t signature[64]);
```

For each of the user's applications, the Python host embeds the application program in a separate sandboxed instance. `friends.c` is the source code for a distributed public key infrastructure application where users produce signed attestations of friendship (similar to that of lab 2):

```
void certify_friend(const uint8_t friend_public_key[32],
                    uint8_t signature[64]) {
  sign(friend_public_key, signature);
}
```

`payment.c` is the source code for an application where users sign transactions for a payment system:

```
void authorize_payment(const uint8_t *payment, uint16_t payment_len,
                       uint8_t signature[64]) {
  uint8_t hash_out[32];
  hash(payment, payment_len, hash_out);
  sign(hash_out, signature);
}
```

The programs above are compiled into `auth.wasm`, `friends.wasm`, and `payment.wasm` before they are instantiated, and the Python host allows the applications to call the `hash` and `sign` functions exported by `auth.wasm`.

Assume the following:

- `private_key` is securely confidential to `auth.wasm`.
- `private_key` is generated exactly once with `key_gen`.
- `hash` implements a collision-resistant hash function.
- `sign` implements a secure public-key signature scheme.

(a) [5 points] How many bytes are copied from the `payment.wasm` sandbox to the `auth.wasm` sandbox memory by the Python host in a single call of `authorize_payment`? Explain your answer.

Solution: 32 (from `sign`) + `payment_len` (from `hash`) bytes. (An answer with an extra 2 bytes on the stack for `payment_len` itself, or an extra 4 bytes for each copied pointer (perhaps with padding for alignment) is also accepted.)

(b) [5 points] Suppose that an attacker can make arbitrary calls to `authorize_payment`. Can the attacker certify a malicious public key as a friend? Explain why or why not.

Solution: No. `hash` is collision-resistant, so it will be intractable for an attacker to get a valid signature for the malicious public key. Since the `sign` is a secure public-key signature scheme, it is infeasible for an attacker to derive a secret key which corresponds to the output of `hash`.

(c) [5 points] Suppose that an attacker can make arbitrary calls to `certify_friend`. Can the attacker authorize a malicious payment? Explain why or why not.

Solution: Yes. Since the private key is the same between the sandboxes, the attacker can call `certify_friend` where the parameter `friend_public_key` is set to be the hash of the malicious payment.

Problem 14. [5 points] **Differential Privacy** (1 part)
Let f be a function over sensitive data sets. The data sets D and D' both have n records. We will define the global sensitivity of f as:

$$S(f) = \max_{\forall dist(D,D')=1} |f(D) - f(D')|$$

where $dist(D, D') = 1$ if and only if D' can be obtained from D by changing *one* record in D.

Any possible record has a single value that is in between the minimum value a and maximum value b.

What is the global sensitivity when f is the median?

In order to get ϵ-differential privacy, what noise distribution should we add to the outcome $f(D)$?

Solution: $b - a$, since the median can change from a to b, or vice versa, in the worst case. Noise distribution that needs to be added has mean $\frac{b-a}{\epsilon} Lap(0, 1)$.

Problem 15. [20 points] **Timing Side Channel** (4 parts)
Recall Lab 5 and the token comparison algorithm that was susceptible to timing attack. Bob now wants to use it to compare passwords.

Note that in this problem, you should ignore timing differences due to micro-architectural structures.

(a) [5 points] Bob thinks he has solved the timing issue. Assuming the real password is of length ≥ 1, he hopes his algorithm respects the following properties:

- **Correctness:** The algorithm will return True if the password provided by the user is the exact same one as expected.
- **Security:** The algorithm will return False if the password is not the one expected.
- **Timing Independence:** The execution timing of the algorithm is independent of the real password.

Look at the following code snippet.

```
def check_password(p, real_p):
  if len(p) != len(real_p):
    return False
  result = True
  for i in range(len(p)):
    result = result and (p[i] == real_p[i])
  return result
```

Can an attacker recover any sensitive information regarding the real password using a timing attack? Can this have an impact on security?

Solution: The attacker can recover the length of the password. This can weaken security as the password is now to be guessed within $0, 1^n$ instead of within $0, 1^*$. Futhermore, some python optimizations might make the "and" operation compute faster once result==False (i.e. not compute the char comparaison)

(b) [5 points] Bob decides to write another version:

```
def check_password(p, real_p):
  max_idx = len(real_p) - 1
  result = True
  for i in range(len(p)):
    j = min(i, max_idx)
    c = real_p[j]
    result = result and (p[i] == c)
  return result
```

Can an attacker recover any sensitive information regarding the real password using a timing attack? Is this code secure as defined in the previous question?

Solution: Yes it is timing independent. No this code is not secure! This code will accept any substring of the password as a valid password even the empty string(!) Additionally, longer passwords can be accepted if they contain the real password as a prefix. For instance "Helloo" being accepted when the real password is "Hello"

(c) [5 points] In the next questions, assume that `check_password` is secure and satisfies timing independence as defined earlier. Bob would like to use it to perform some access control into his secure server. He hopes his scheme respects the following properties:

- **Correctness:** The algorithm will return True if the given username correspond to an existing user on the server and the associated password is the exact same one as provided by the user.
- **Security:** The algorithm will return False if the username is not registered on that machine or the password is not the one associated with the given username.
- **Timing Independence:** The execution timing of the algorithm should not help a potential attacker to recover *any* information regarding other users of the server.

He writes the following code:

```
def has_access(username, password):
  if username not in user_password_dict:
    return False
  else:
    return check_password(password, user_password_dict[username])
```

Can an attacker recover any sensitive information regarding the secure server using a timing attack? Can this have an impact on security?

Solution: Yes: if a given username is used or not by the server. This can help the attacker to only perform brute-force attack to guess the password of existing users.

(d) [5 points] Bob decides to write another version:

```
def has_access(username, password):
  cond = username not in user_password_dict
  empty_password = '00000000'
  if cond:
    real_p = empty_password
  else:
    real_p = user_password_dict[username]
  return check_password(password, real_p)
```

This code has a series of security and timing issues. Detail one of them.

Solution: 1) The code returns True for any request that associate a username that is not in the user base and the password '00000000' 2) The code performs a dict access if the username is real and no access if not. That might still leak timing information.

Problem 16. [10 points] **Course Survey** (4 parts)
This is the first time we taught this class, and we would like your feedback on how to improve this class when we teach it next year. **Any answer, except a blank answer, will receive full credit.**

(a) [3 points] How did this class match up with your expectations for what you wanted to learn by taking the class?

(b) [3 points] How important to you are the breadth of covered topics (authentication, encryption, systems, software, hardware, privacy, human factors, etc); the depth in each of the topics; and the connection between the topics? Which of these needs the most improvement the next time we teach this class?

(c) [2 points] How did taking this class affect your thinking about taking other security-related classes such as 6.857, 6.858, and 6.875?

(d) [2 points] What was your favorite part of the lab assignments that we should keep, and what was the most tedious / least favorite aspect that we should fix?