You are to work on the following questions *alone*. Do not discuss these questions with anyone. Typeset your answers and submit as a `PDF`.

1. Answer the following questions about the ABD multi-writer, multi-reader register algorithm, presented in Algorithm 1.
   (a) (5 points) Recall Lamport's safe, regular, and atomic register semantics. The ABD algorithm guarantees atomicity. In this algorithm, both reads and writes consist of two phases, which for the purposes of this question we'll refer to as the query phase and the store phase. Suppose we only wanted to guaranteed *regular* semantics. How could the protocol be made more efficient?
   (b) (2 points) Now suppose that we do want atomic register semantics, as in the original protocol. What additional checks could you add to the ABD protocol to take advantage of the optimization you made in part (a) in the "common" case?
   (c) (8 points) Suppose the ABD algorithm is modified such that a write is sent speculatively in the query phase (with a new timestamp) and that the store phase is skipped if the speculative write succeeded at a majority. The modified algorithm is shown in Algorithm 2. The QUERY message handler on the server and the COMMUNICATE function in the client were modified, and a field was added to the client. Is this protocol safe; does it still provide atomicity? If it does, briefly explain why. If not, provide a counter-example trace that demonstrates the problem.
2. Answer the following questions about the BFT state machine replication algorithm presented in the "Practical Byzantine Fault Tolerance" paper.
   (a) (2 points) Can a client spoof a request such that it appears to have been initiated by a different client? Briefly justify your answer.
   (b) (2 points) Consider a faulty leader that ignores client requests. How does the algorithm make progress as long as there are a sufficient number $(2f + 1)$ of non-faulty replicas (and sufficient network synchrony)?
   (c) (2 points) Consider a faulty leader that assigns different sequence numbers to the same client command in its pre-prepare messages to other replicas. How does the algorithm ensure that a client command isn't executed at different positions in the command log?
   (d) (2 points) Assume that you have a faulty replica that received a pre-prepare message assigning sequence number n to client request m. What happens if the replica sends a prepare message associating a different sequence number n' to the client request?
   (e) (2 points) Assume that you have a faulty replica. What happens when the replica sends an incorrect response to the client?
   (f) (5 points) The paper describes an alternative method to message authentication using what they call *authenticators*. First, each pair of servers sets up a shared key at the beginning of the execution. A message authentication code (MAC) for a message sent from s_1 to s_2 is a string generated from the message text and the shared key between s_1 and s_2 that

Algorithm 1 ABD MRMW Atomic Register Algorithm

Server local state:

$t \leftarrow (0, \perp)$ ▷ Current timestamp, initially unique minimum value; lexicographically ordered

$v \leftarrow \perp$ ▷ Current value, initially special null value

1: **upon** receiving $\langle$QUERY$\rangle$
2: Send reply $\langle$QUERY-REPLY$, t, v\rangle$ ▷ Messaging infrastructure correctly associates messages with replies
3: **end upon**
4: **upon** receiving $\langle$STORE$, t', v'\rangle$
5: **if** $t < t'$ **then**
6: $t \leftarrow t'$
7: $v \leftarrow v'$
8: **end if**
9: Send reply $\langle$STORE-REPLY$\rangle$ ▷ Messaging infrastructure correctly associates messages with replies
10: **end upon**

Client local state:

p ▷ Unique process ID, immutable

11: **procedure** READ
12: COMMUNICATE($\perp$)
13: **end procedure**
14: **procedure** WRITE(v)
15: COMMUNICATE(v)
16: **end procedure**
17: **function** COMMUNICATE(v)
18: Send $\langle$QUERY$\rangle$ to all servers
19: Wait for $\lfloor \frac{n}{2} \rfloor + 1$ replies, stored in R
20: $t \leftarrow \max\{m.t : m \in R\}$ ▷ Maximum timestamp out of replies seen
21: **if** $v = \perp$ **then**
22: $v \leftarrow m.v : m \in R \wedge m.t = t$ ▷ Value associated with maximum timestamp
23: **else**
24: $t \leftarrow (t[0] + 1, p)$
25: **end if**
26: Send $\langle$STORE$, t, v\rangle$ to all servers
27: Wait for $\lfloor \frac{n}{2} \rfloor + 1$ replies
28: **end function**

Algorithm 2 Modified MRMW Register Algorithm

Server local state:
$t \leftarrow (0, \perp)$
$v \leftarrow \perp$

1: **upon** receiving $\langle \text{QUERY}, t', v' \rangle$
2: **if** $v' \neq \perp \wedge t < t'$ **then**
3: $t \leftarrow t'$
4: $v \leftarrow v'$
5: **end if**
6: Send reply $\langle \text{QUERY-REPLY}, t, v \rangle$
7: **end upon**

8: **upon** receiving $\langle \text{STORE}, t', v' \rangle$
9: **if** $t < t'$ **then**
10: $t \leftarrow t'$
11: $v \leftarrow v'$
12: **end if**
13: Send reply $\langle \text{STORE-REPLY} \rangle$
14: **end upon**

Client local state:
p
$t \leftarrow (0, \perp)$ ▷ Local timestamp

15: **procedure** READ
16: COMMUNICATE($\perp$)
17: **end procedure**

18: **procedure** WRITE(v)
19: COMMUNICATE(v)
20: **end procedure**

21: **function** COMMUNICATE(v)
22: **if** $v \neq \perp$ **then**
23: $t \leftarrow (t[0] + 1, p)$
24: **end if**
25: Send $\langle \text{QUERY}, t, v \rangle$ to all servers
26: Wait for $\lfloor \frac{n}{2} \rfloor + 1$ replies, stored in R
27: $t' \leftarrow \max\{m.t : m \in R\}$
28: **if** $v \neq \perp \wedge t' = t$ **then** ▷ Speculative write succeeded at a majority
29: **return**
30: **else if** $v \neq \perp$ **then**
31: $t \leftarrow (t'[0] + 1, p)$
32: $t' \leftarrow t$
33: **else**
34: $v \leftarrow m.v : m \in R \wedge m.t = t'$
35: **end if**
36: Send $\langle \text{STORE}, t', v \rangle$ to all servers
37: Wait for $\lfloor \frac{n}{2} \rfloor + 1$ replies
38: **end function**

proves that the sender of the message knew the shared key. An authenticator for a message broadcast by s_1 is a vector of MACs, one for each of the message's recipients.

Suppose that checkpoint messages contained authenticators, rather than digital signatures, and that servers only wait for $f+1$ matching checkpoints before garbage collection. ($f+1$ messages were sufficient when digital signatures were attached to checkpoint messages.[1]) What could go wrong?

3. (10 points) Consider a BigTable client whose cache is empty, i.e., it has never talked to a given BigTable deployment before. Enumerate the requests it will make to read a single row with key k from table T.

4. (10 points) Give one scenario where GFS's "record append" would insert duplicate records at the end of a file.

5. Answer the following about the Bitcoin protocol.

   (a) (5 points) Suppose that an attacker gains control of $> 50\%$ of the compute power currently operating on the Bitcoin network. Could this attacker steal money from arbitrary victims (who have had no previous transactions with the attacker) in such a way that non-faulty observers will acknowledge the fraudulent transactions? How or why not?

   (b) (5 points) Miners refuse to add invalid transactions (including transactions whose inputs have already been spent) to their proposed block. What incentive does a miner have to follow this procedure? Why doesn't it just add the transaction to the block and let consumers of the transaction log ignore invalid transactions at playback time?

[1] Refer to the lecture slides and the PBFT Technical Report. The original paper incorrectly implies that $2f+1$ signed checkpoint messages are needed.