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

1. Answer the following questions about the ABD multi-writer, multi-reader register algorithm, presented in Algorithm 1.

   (a) (6 points) Recall Lamport's safe, regular, and atomic register semantics. These were defined in terms of single-write registers, but there is a natural way to extend these definitions to multiple writers.

   The ABD algorithm guarantees atomicity (linearizability). 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.

   In the multi-write case, regularity implies that for some real-time respecting ordering of writes, reads can return either the value written by the most recently completed write (in the sequential ordering) or any of the values currently being written.

   Suppose we only wanted to guarantee regular semantics as defined above. How could the protocol be made more efficient?

   (b) (4 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. Does this protocol 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 sends different pre-prepare messages to different nodes for the same slot, i.e., it tells some nodes that it has assigned a client command c_1 to a given slot and tells other nodes that it has assigned a client command c_2 to the same slot. How does the algorithm deal with this issue?

   (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?

Algorithm 1 ABD MRMW Atomic Register Algorithm

```
Server local state:
    t ← (0, ⊥)        ▷ Current timestamp, initially unique minimum value; lexicographically ordered
    v ← ⊥             ▷ Current value, initially special null value

upon receiving ⟨QUERY⟩
    Send reply ⟨QUERY-REPLY, t, v⟩     ▷ Messaging infrastructure correctly associates messages with replies
end upon

upon receiving ⟨STORE, t′, v′⟩
    if t < t′ then
        t ← t′
        v ← v′
    end if
    Send reply ⟨STORE-REPLY⟩     ▷ Messaging infrastructure correctly associates messages with replies
end upon
```

```
Client local state:
    p                 ▷ Unique process ID, immutable

procedure READ
    COMMUNICATE(⊥)
end procedure

procedure WRITE(v)
    COMMUNICATE(v)
end procedure

function COMMUNICATE(v)
    Send ⟨QUERY⟩ to all servers
    Wait for ⌊n/2⌋ + 1 replies, stored in R
    t ← max{m.t : m ∈ R}                ▷ Maximum timestamp out of replies seen
    if v = ⊥ then
        v ← m.v : m ∈ R ∧ m.t = t        ▷ Value associated with maximum timestamp
    else
        t ← (t[0] + 1, p)
    end if
    Send ⟨STORE, t, v⟩ to all servers
    Wait for ⌊n/2⌋ + 1 replies
end function
```

Algorithm 2 Modified MRMW Register Algorithm

```
Server local state:
    t ← (0, ⊥)
    v ← ⊥

upon receiving ⟨QUERY, t′, v′⟩
    if v′ ≠ ⊥ ∧ t < t′ then
        t ← t′
        v ← v′
    end if
    Send reply ⟨QUERY-REPLY, t, v⟩
end upon
upon receiving ⟨STORE, t′, v′⟩
    if t < t′ then
        t ← t′
        v ← v′
    end if
    Send reply ⟨STORE-REPLY⟩
end upon
```

```
Client local state:
    p
    t ← (0, ⊥)                                   ▷ Local timestamp

procedure READ
    COMMUNICATE(⊥)
end procedure
procedure WRITE(v)
    COMMUNICATE(v)
end procedure
function COMMUNICATE(v)
    if v ≠ ⊥ then
        t ← (t[0] + 1, p)
    end if
    Send ⟨QUERY, t, v⟩ to all servers
    Wait for ⌊n/2⌋ + 1 replies, stored in R
    t′ ← max{m.t : m ∈ R}
    if v ≠ ⊥ ∧ t′ = t then                       ▷ Speculative write succeeded at a majority
        return
    else if v ≠ ⊥ then
        t ← (t′[0] + 1, p)
        t′ ← t
    else
        v ← m.v : m ∈ R ∧ m.t = t′
    end if
    Send ⟨STORE, t′, v⟩ to all servers
    Wait for ⌊n/2⌋ + 1 replies
end function
```

(e) (2 points) Assume that you have a faulty replica. What happens when the replica sends an incorrect response to the client?

(f) (6 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 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.) What could go wrong?

3. (8 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. (8 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. This is optional credit.

   (a) (5 points) Suppose that an attacker gains control of $> 50\%$ of the compute power currently operating on the Bitcoin network. Could this attacker convince non-faulty observers that improperly signed transactions are actually valid?

   (b) (5 points) Suppose the difficulty were decreased dramatically so that the average rate of block discovery was much higher than once every ten minutes. Further suppose that clients proportionally increased the number of blocks deep they waited for a transaction to be considered confirmed. What could go wrong?

   (c) (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?