CS251: Cryptocurrencies and Blockchain Technologies **Fall 2022**

Assignment #2

Due: 11:59pm on Tue., **Oct. 25, 2022**
Submit via Gradescope (each answer on a separate page) code: **DJ66V3**

Problem 1. Why is the difficulty of the proof of work in Bitcoin set to ten minutes? What would go wrong if it were changed to ten seconds?

Problem 2. Suppose two groups independently implement the Bitcoin protocol. Some miners run implementation A and other miners run implementation B. At some point an attacker finds a vulnerability in implementation A that causes miners running that implementation to accept transactions that double spend a UTXO. Implementation B treats such transactions as invalid.

a. Suppose 80% of the mining power runs the buggy implementation and 20% runs the non-buggy one. What will happen to the blockchain once a block containing a double-spending transaction is submitted to the network?

b. What will happen to the blockchain in the reverse situation where 20% of the mining power runs the buggy implementation and 80% runs the non-buggy one? That is, what will happen to the blockchain once a block containing a double-spending transaction is submitted to the network?

Problem 3. Streamlet. In Lecture 4 we discussed the Teen-StreamLet protocol. We proved (slide 30) that if the adversary corrupts fewer than $n/3$ replicas (i.e., $f < n/3$) then for every epoch e, there can be at most one notarized block with epoch number e, in the view of any honest replica. Suppose we change the notarization rule (slide 26) as follows:

> A block B is said to be notarized in the view of a replica, if the replica observes over $\mathbf{3n/4}$ signatures from distinct replicas on the block B.

a. What is the largest number of replicas that the adversary can control so that the Lemma on slide 30 remains true? That is, what is the largest value f for which the lemma still holds?

b. It seems that this new notarization rule makes the protocol more robust: the lemma on slide 30 remains true for a larger value of f than in teen-StreamLet. Why did Teen-StreamLet use $2n/3$ as the notarization threshold? To answer the question, explain how many replicas the adversary needs to control under the new rule to violate liveness? Note that in Teen Streamlet liveness is violated if no block can be notarized. Is liveness better or worse than in Teen-Streamlet?

Problem 4. Reliable broadcast. Consider n parties, where $n \geq 3$, and where one of the parties is designated as a *sender*. The *sender* has a bit $b \in \{0, 1\}$. A *broadcast protocol* is a protocol where the parties send messages to one another, and eventually every party outputs a bit b_i, for $i = 1, \ldots, n$, or outputs nothing.

- We say that the protocol has **safety** if for every two honest parties, if one party outputs b and the other outputs b', then $b = b'$.
- We say that the protocol has **validity** if when the *sender* is honest, the output of all honest parties is equal to the *sender*'s input bit b.
- We say that the protocol has **totality** if whenever some honest party outputs a bit, then eventually all honest parties output a bit.

A *reliable broadcast protocol* (RBC) is a broadcast protocol that satisfies all three properties.

Let us assume that there is a public key infrastructure (PKI), meaning that every party has a secret signing key, and every party knows the correct public signature verification key for every other party.

In a synchronous network, consider the following broadcast protocol:

- *step 0:* The *sender* sends its input bit b (along with its signature) to all other parties. The *sender* then outputs its bit b and terminates.
- *step 1:* Every non-sender party i echoes what it heard from the *sender* to all the other non-sender parties (with i's signature added). If the party heard nothing from the *sender*, it does nothing in this step. Similarly, the party does nothing in this step if the *sender*'s message is malformed: for example, if the *sender*'s signature is invalid, or the message is not a single bit.
- *step 2:* Every non-sender party collects all the messages it received (up to $n - 1$ messages, with at most one from the *sender* in step 0 and at most one from each non-sender party in step 1). If some two of the received messages contain a valid signature by the *sender*, but for opposite bits (i.e., in one signed message the bit is 0 and in the other signed message the bit is 1) then the *sender* is dishonest and the party outputs 0 and terminates. Otherwise, all the properly signed bits from the *sender* are the same, and the party outputs that bit. If the non-sender received no messages, it outputs nothing.

For each of the following questions, describe an attack or explain why there is no attack.

a. If there is at most one dishonest party, does the protocol have safety?

b. If there is at most one dishonest party, does the protocol have validity?

c. If there are at most two dishonest parties, show that the protocol does not have safety.

d. If there are at most two dishonest parties, does the protocol have validity?

e. Does the protocol have totality (for any number of dishonest parties)?

Problem 5. Synchronizing the mempool across validators. Validators A and B each have a set of transactions in their mempool. Suppose that validator A's set is a superset of validator B's. Validator A wants to send to B the transactions that B is missing. The problem is that A does not know which transactions B is missing.

a. Suppose B is only missing one transaction. Show that A can send a single 32-byte message to B that quickly lets B identify the missing transaction hash. B will send the missing transaction hash to A, and A will send back the transaction data.

Hint: Think of computing the xor of all the transaction hashes in A's mempool.

b. Suppose B is missing k transactions, for some small k. There is a simple algebraic solution for sending the k missing transactions to B, but here we will look at an interactive solution instead. Show that A and B can engage in a protocol that takes at most 256 rounds to identify the k missing transaction hashes (every hash is 32 bytes or 256 bits long). In every round A sends up to $2k$ hash values to B, and B responds with up to $2k$ bits. In the last round, B can send up to k hashes to A.

We would like the protocol to terminate in the fewest number of rounds. Suppose transaction hashes are uniformly distributed in the hashing range. What is the expected number of rounds until your protocol terminates, in terms of n, the total number of hashes that A has?

Hint: In the first round try applying the method from part (**a.**) twice – once to A's transactions in the lower half of the hash range and once to A's transactions in the upper half.

Problem 6. Bob posts the following wallet contract to Ethereum to manage his personal finances:

```
contract BobWallet {
    function pay(address dest, uint amount) {
       if (tx.origin == HardcodedBobAddress) dest.send(amount);
} }
```

The function `pay` lets Bob send funds to anyone he wants. Suppose Mallory can trick Bob into calling a method on a contract she controls. Explain how Mallory can transfer all the funds out of Bob's wallet into her own account.

Hint: Make sure you understand the semantics of `tx.origin`.