CS251: Cryptocurrencies and Blockchain Technologies **Fall 2023**

Assignment #2

Due: 11:59pm on Tue., **Oct. 24, 2023**
Submit via Gradescope (each answer on a separate page) code: **7DVJKY**

Problem 1. 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 2. In lecture 5 we discussed Nakamoto's private attack (starting on slide 22). The attack shows that if the adversary controls more than half the miners (or more accurately, more than half the hashing power) then they can cause, say, one year worth of transactions to be deleted from the tip of the chain and replaced with one year of transactions that effectively do nothing (for example, a sequence of blocks that only contain the coinbase Tx and nothing else). This means that everyone who was paid during the year loses their income; the UTXOs holding their assets simply disappear and the funds go back to the UTXOs from which they came. Recall that the fraction of the hashing power controlled by the adversary at time t is denoted by $\beta(t) \in [0, 1]$.

a. Suppose $\beta(t)$ is equal to exactly $1/2$ for all t. Can the attacker still replace about a year worth of transactions with transactions that effectively do nothing?

b. Explain why when $\beta(t) = 0.2$ for all t, the attack will not work, with high probability.

Problem 3. Consider again the simple PoS consensus protocol from Lecture 6 (starting on slide 12). Suppose there are n validators in the system. The confirmation rule we discussed in the lecture says that once at least $\mathbf{2n/3}$ of the validators vote for the block (i.e., they sign the block using their secret signing key), the block is confirmed. We argued that if two blocks are confirmed at a single position in the chain then at least $n/3$ validators must be malicious. Therefore, if there are fewer than $n/3$ malicious validators, then safety is maintained.

a. Suppose we change the confirmation rule so that a block is confirmed once at least $\mathbf{3n/4}$ of the validators vote for the block. What is the upper bound on the number of malicious validators (as a function of n) for safety to be maintained?

b. In part (a) you showed that the modified protocol ensures safety against more malicious validators than the protocol from the lecture. This comes at the cost of liveness. In the

modified protocol, explain why $(n/4)+1$ malicious validators can cause the chain to halt and confirm no new blocks. Recall that the protocol from the lecture guaranteed liveness up to $n/3$ malicious validators.

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 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. Recall that a dishonest party (or parties) does not have to follow the protocol and can send whatever messages it wants, or perhaps not send a message at all.

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. 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`.