CS 5433: Blockchains, Cryptocurrencies, and Smart Contracts

Homework 1

jas2248@cornell.edu
jeh422@cornell.edu
dm767@cornell.edu

1. Hybrid Systems

1d. There are a variety of system designs possible beyond the proof of work and authority sealing mechanisms we've explored here. Name one security problem with the proof of authority sealing mechanism from (1b). Can you think of a technically sound approach that could help mitigate this problem (imperfect mitigations are fine)? (Please do not describe existing protocols, including but not limited to Proof of Stake protocols, Tendermint, or PBFT-based systems; originality is highly valued here!).

The problem for this kind of proof authority, in which the corresponding private key available only to the trusted authority (the validator), puts in a high risk the security of the network by concentrating the validation power to one node. If an attacker is able to hack the validation node and perpetrate an attack the network will fail. One solution that we proposed is that the trusted authority can create aliases of himself that are known to participants. Each of these aliases are able to validate transactions randomly (but verifiable for honest nodes), while maintaining the capacity to audit the actions of this party and its aliases. Another solution that we propose is a MULTISIG for the validation of transaction, in this sense the verification process will not depend on only one node but in multiple nodes. All this is intended for increasing the security of the network by decreasing the vulnerability of the validation node.

2. UTXO Management in Wallets

Overview of the problem and assumptions

- We assume that Moonbase is a bitcoin exchange-wallet system that allows to do transactions over the bitcoin blockchain.
- We assume that the Moonbase business model take advantage of networks effects (more value when more users are on the platform). For this reason, we assume that Moonbase exchange-wallet business model relies on having a large pool of UTXO for all of its users. This means that they have a “complete” spectrum of UTXOs values that can be used for transactions and they can have advantage by using the exact UTXO needed. They create profits by using the pool of UTXOs transactions and incurring in less networks fees.
- We assume that the only income Moonbase has is the network fee and no other one.
- We assume that when the user “withdraws” they actually are spending his bitcoin assets. Moonbase take a random UTXO from the large spectrum pool that can satisfy this transaction (has to be bigger than the withdraw).

2.1 Identify a denial-of-service vector in this process, i.e., an adversarial attack strategy by dishonest and wealthy users that could result in failed withdrawals for legitimate users of Moonbase. Provide a fix for this DoS vector and argue informally that user withdrawals will never fail.

We assume the attacker has access to Moonsbase UTXO full set values.
We assume that an attacker (master node) can orchestrate an attack by requesting a high amount of payment transaction of equal very small sizes (what is called *dust)* to an address that the attacker controls.
With these two assumptions the attacker can start requesting UTXO withdrawals of high UTXO values (because they know what are the UTXO highest values for Moonbase), leaving Moonbase with very small size UTXOs that were created by the attack. This will create a denial-of-service for legitimate users that might request withdrawals with for high transactions because there are not going to be UTXOs that can complete their transactions.
Not only this attack might create a denial-of-service, which Moonbase could circumvent by generating exact UTXO’s for the specific request by having a large UTXO (reserve) where they can rely on, but by doing this, Moonbase will incur in high network fees for providing its services for legitimate users, eventually making an unprofitable service.
One way to fix this, is that Moonbase could use an algorithm to maintain a large pool of high UTXO values, so with this they can protect themselves for not having enough UTXO to provide payment services. Another way to prevent this is to limit the amount of withdrawals/payments by a certain threshold of your contribution to the blockchain, maybe limit to 100x of the value you have transacted. With this you can prevent attacks orchestrated by small users with small very small transactions to overflow the network.

2.2 For each user withdrawal in the provided scheme, recall from class that two UTXOs actually need to be generated: one paying the target user, and one that is kept by Moonbase representing any leftover “change”. Provide a modi- fication to the above strategy that will reduce the number of UTXOs Moonbase must maintain in its database.

They can create a matching-optimization algorithm that matches a “normal” UTXO with a “dust” UTXO to get rid of the smallest UTXOs in their database. With this algorithm you can ensure a combination of UTXOs that decrease the cost of consolidating a payment rather than create more UTXO dust.

3. Consensus

3.1 Under what corruption threshold does the protocol satisfy validity? For instance, does it satisfy validity when < n/6, < n/4 or even < n/3 of the players are faulty? Prove it, and provide an attack showing that your bound is optimal. (Recall that providing validity means that if the sender is honest, then all honest players need to output the sender's input.)

The protocol satisfies validity for corruption threshold < $\frac{1}{4}$ n. Let's prove by contradiction that this bound is optimal.
Assume there are more than $\frac{1}{4}$ n faulty players and our protocol satisfies validity. Here all the faulty players can collude and and not vote for the correct bit m. Then, we will have a maximum votes of less than (n - $\frac{1}{4}$ n) for the correct bit m, which is $\frac{3}{4}$ n. Since there is no majority of $\frac{3}{4}$ n votes, according to the round 3 protocol, the players (including the honest) will output ⊥. Since the honest players' response ⊥ is different from m, this violates validity ad hence the contradiction. This proves that the threshold is < $\frac{1}{4}$ n.

3.2 Show that the protocol does not satisfy consistency even if just the sender is faulty. Provide an explicit attack. (Recall that breaking consistency means coming up with an attack that leads 2 honest players to output different values; for instance, one of the could output some message m, and a different one outputs ⊥)

Let's us imagine an n-party with 1 faulty sender and n-1 honest receivers.

<u>Assumptions:</u>
While tallying the number of votes in round-3, receivers:

1. consider the sender's initial multicasting in round-1 as a vote
2. consider their own vote (same as the sender's message if the receiver is honest)

<u>Round 1:</u>
Assume that the sender initially multicasts the correct bit m to exactly $\frac{3}{4}$ n - 1 other receivers and a different faulty message to the other $\frac{1}{4}$ n receivers. Pick two receivers P_i who received and P_j who received a faulty message.

<u>Round 2:</u>
Since all the n-1 players (other than sender) are honest, they will multicast (or vote) the same message they received. This will involve $\frac{3}{4}$ n - 1 players voting for the correct bit m and others passing on their faulty messages they received from the sender.
However, since the sender is faulty, she can vote anything to anyone. Assume that the sender votes the correct bit m to P_i but a faulty bit m' to P_j

Round3:
Player P_i received $\frac{3}{4}n$ votes of bit m (including sender), so will output m
Player P_j received only $\frac{3}{4}n - 1$ votes of bit m (since sender sent a faulty message), so she will output $\perp$

Here is a contradiction, as we found two honest players P_i and P_j with different outputs, failing the consistency.

3.3 Bonus: Show that the protocol satisfies a weaker form of consistency where all honest players either output the same message m, or they output $\perp$, even if < n/2 of the players are faulty. (That is, you need to show that if 2 honest players output messages m1, m2, neither of which is $\perp$, then m1 = m2.) Hint: prove a variant of the quorum intersection lemma shown in class.

Assume for contradiction that m1≠m2.

Note there are faulty players less than n/2, so $|F| < \frac{1}{2}n$

Let S_1 be the set of players that vote m1, and
Let S_2 be the set of players that vote m2

Since, the players need atleast $\frac{3}{4}n$ votes in order to output a bit, we have

$$|S_1| \geq \frac{3}{4}n$$
$$|S_2| \geq \frac{3}{4}n$$
$$|S_1| + |S_2| \geq \frac{3}{4}n + \frac{3}{4}n = \frac{3}{2}n$$

Recall that honest players vote for a unique bit per round, thus no honest player can be in both S_1 and S_2, thus can contribute at most n - |F| (i.e., the number of honest players) to the above sum. Faulty player, however, can send different bits to sets 1 and 2 and thus be in both set and can thus contribute at most 2|F| to this sum. We thus also have that,

$$|S_1| + |S_2| \leq n - |F| + 2|F| = n + |F| < \frac{3}{2}n$$

which is a contradiction

Evaluation

Did you find the homework easy, appropriately difficult, or too difficult?
Appropriately difficult for the course. While we had a good understanding on the overall coding aspects, since none of us had a full time software coding experience, we ended up spending a lot more time in fixing even minor bugs.

How many hours total (excluding breaks :)) were spent on the completion of this assignment?
About 20-22 hours. Theoritical questions took only about 3-4 hours, while the coding part took the rest of the time.

Did you feel there was too much coding, the appropriate amount of coding, or not enough coding?
We felt the coding part was slightly on the higher side