CS 5433 Blockchains, Cryptocurrencies, and Smart Contracts

Homework 1

Yan Jiang yj334 | Carolina Peisch cgp48 | Disheng Zheng dz336

Problem 1 - Signatures, Hashes, Sealing [40]

(1c) - Hybrid Systems

One problematic security aspect of proof of authority sealing mechanisms is that because validators must have their identity formally verified to be approved, there is no anonymity at all. While this might initially seem like a safer option than other systems, as there is more transparency and ownership to each validator and their work, this also means that such validators know who each other are and could theoretically find each other and create a system in which they are manipulating the transactions. The PoA system of attaching reputation to a validator's identity only prevents one single validator from diverging, but if enough validators join together and agree to ignore the risks of damaged reputation, this incentive to behave fairly and justly no longer exists.

Therefore, one approach that could mitigate this problem would be PseudoPoA. PseudoPoA would be a mechanism similar to traditional PoA but with one major difference – validators would never reveal their actual identity, and instead base this same system of reputation on a pseudonym. This system would be stricter than a public network, and validators would not be able to create multiple identities as they wished, but they would be limited to one single, pseudonymous identity. It would be impossible to trace a validator's pseudonym to their true identity, and this atmosphere of uncertainty on true identities, paired with the presence of PoliceVals, would seriously deter any sort of collusion. PoliceVals would be embedded validators in the network which would appear to be normal validators, until an adversarial validator approached them trying to collude, in which case the PoliceVals would eject them from the network and they would be forever banned from entering. In conclusion, PseudoPoA would still benefit from attaching pseudo-identity to validator reputation, albeit a weaker incentive now that only a pseudonym (yet still

immutable) reputation is a stake, but PseudoPoA offers a stronger system against validator collusion and related transaction manipulation.

References:

https://en.wikipedia.org/wiki/Proof-of-authority

https://ethereum.stackexchange.com/questions/13736/what-are-the-limitations-of-proof-of- authority/19457#19457

Problem 2 - UTXO Management in Wallets [15]

2.1

One adversarial attack strategy could be if the dishonest and wealthy users banded together (they can communicate between each other) and sent massive amounts of transactions consisting of very small amounts to flood the block maximum. Since they are wealthy, they could easily cover the transaction costs. This enormous amount of resulting UTXOs, all representing tiny amounts, would mean that legitimate users of Moonbase would be prevented from withdrawing their funds because there would be way too many UTXOs of small amounts that wouldn't be large enough to fund any of their transactions. One potential fix would be a floor minimum for the amount a user could transact, rendering the playing field fairer to those users who are using the platform honestly and legitimately. Therefore, the adversarial attackers wouldn't be able to send tiny amounts (such as 1 cent) to flood the block.

Another attack strategy would be for a very wealthy group of users to manipulate the transaction fee and raise it to a high amount that would preclude normal users from affording such fees. If I wanted to pay a transaction fee of 10x the usual amount, my transaction is more likely to go through. If I continued to pay this 10x transaction fee, the other users wouldn't get their transactions through, as they are only willing to pay a fraction of this. A fix to this situation would be a set, immutable transaction fee cost that wouldn't allow these wealthy bad guys to manipulate and increase the transaction fee to exclude the legitimate user withdrawals.

2.2

One simple modification would be to consolidate all one single user's UTXOs so that Moonbase wouldn't have multiple UTXOs per user based on their number of transactions, but instead one single UTXO per user. Therefore, if Alice has 10 Bitcoin and sends Bob 5 Bitcoin, one UTXO would go to Bob (5 Bitcoin), and the other would be stored by Moonbase (5 Bitcoin). If Alice sent Carl 3 Bitcoin, then one UTXO would go to Carl (3 Bitcoin), and Moonbase would add this new UTXO of 2 Bitcoin to Alice's first UTXO, giving her one single UTXO that Moonbase must maintain. This system will also have the benefit of ensuring that not all UTXOs will represent a tiny value, which is problematic when a UTXO has to be randomly selected to fund another user's transactions and there are not enough large UTXOs. In addition, not all users will be making the same volume of transactions, meaning that there will still be some users with small-valued UTXOs available for similarly small withdrawals.

Alternatively, Moonbase could require that all users consolidate their UTXOs themselves on a regular basis (say monthly). In this case, a user would collect all their UTXOs consisting of change and sent it to themselves in one single transaction. This would create one single output for any future transactions, and Moonbase wouldn't have to maintain many tiny UTXOs per person.

References:

https://www.youtube.com/watch?v=IehHwBt3NAg
https://medium.com/@lopp/the-challenges-of-optimizing-unspent-output-selection-a3e5d05d13ef
https://www.reddit.com/r/Bitcoin/comments/7rxiwh/few_thoughts_about_utxos_consolidation/

Problem 3 - Consensus [30]

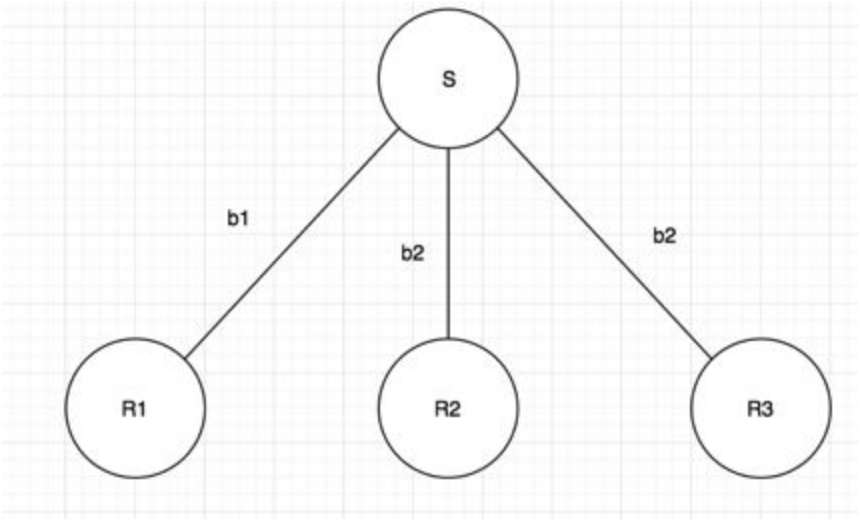
3.1

The corruption threshold that would satisfy validity is n/4 . Since by round 3, each player i checks whether the received some message m from at least 3n/4, if not, the will output _|_.

Here's an attack. Let sender be honest and sent a message m. If F, the set of faulty players, is larger than n/4 , and they output some message m', then all the honest i would see there are less than 3n/4 output m, they will output _|_, which contradicts validity.

3.2

The protocol does not satisfy consistency. Let's consider a scenario where there are 4 players and only the sender is faulty, meaning the sender S can send different values to different players.



Including the votes themselves. Player 1 would see there are 2 votes for b1 and 2 votes for b2 and therefore output _|_. And player 2 and 3 will see there are 1 player vote b1, and 3 players votes b2, which is greater and equal to 3n/4 , and therefore output b2. In the end b2 is not equal to _|_, 2 honest players output different values, thus contradicts consistency.

3.3 Bonus

The protocol satisfies the weak form of consistency.

Assume < n/2 players are faulty. If in some round r, honest player i, j sees 3n/4 votes for bi and bj. Then bi = bj.

Proof by contradiction: We assume bi ≠ bj. Let Si be the set of players send player i a vote for bi. We define Sj analogously. By assumption:

$$|Si| + |Sj| \geq 3n/4 + 3n/4 = 3n/2$$

Recall that honest players vote a unique bit each time. No honest players can be in both Si and Sj, thus they can contribute at most n - |F| to above sum. Faulty players however, can be in both Si, an Sj, thus contribute at most 2|F|.

$$|Si| + |Sj| \leq n - |F| + 2|F| = n + |F| < n + n/2 = 3n/2$$

Which is a contradiction. So bi = bj complies with consistency.

(BONUS) - Merkle Trees

The idea of a Merkle tree is to hash transactions individually to give their corresponding value. After each transaction has been individually hashed to produce its corresponding hash value, the new hash values are combined with an adjacent partner to be hashed once again. This process is repeated until the last hash value is obtained as a Merkle Root. And the Merkle Root is to be incorporated as the block header.

References:

https://www.mycryptopedia.com/merkle-tree-merkle-root-explained/

https://gist.github.com/brandomr/f5f325a0e9161d481714efe00d846880

Evaluation

We found 1a to be very time consuming but the coding itself wasn't too hard. It took a very long time to understand the skeleton code, but once you understood the framework then it was straightforward to proceed.

Question 2 was appropriately difficult.

Question 3 was appropriately difficult, but the wording of the question was really confusing, and it was hard to understand what the question was truly asking. A simpler, more pared down version would have been preferable.

Question 4 was appropriately difficult.

We're not sure how many hours we spent on this assignment, but we agree that it felt like an appropriate amount. The amount of coding was good, having a mix of coding and written questions was good for our team which had members of varying coding levels.