Morgan Locks – tml3cf

Bitcoin PS2

1. Satoshi assumes:
    a. Bitcoin will lose value if an attacker demonstrates the ability to double spend.
    b. The greedy attacker cannot cash out into another currency faster than value is lost.
    c. The mining rewards and block rewards are large enough that the greedy attacker wouldn't just lose money by playing fair. If they would, it may be in the attackers interest to mine greedily although it would devalue bitcoin in the long term, because they lose either way in the long term.
2. "Once a transaction has 340 confirming blocks, the probability that a selfish miner with 45% of the hashing power could overtake the honest blockchain and change that transaction is less than .1%."
3. .

q	z
.1	3
.15	3
.2	5
.25	6
.3	10
.35	17
.4	36
.45	137

4.
    a. When we query for data[i], the remote server could give us a different data[j] in response. The signature would be valid, so we know that they gave us *some* valid piece of data, but we would have no way of knowing that it was data[i].
    b. We can't be sure. If there was an old version of data[i], and we wrote a new data[i], the server could still give us the old data[i] and we would not be able to tell it wasn't the most recent one.
5.
    a.
        i. To write data[i], you must first read all data blocks, verify that they are valid, compute the new hash by replacing the old data[i] with the new data[i], and then write data[i] to the remote server.
        ii. To read data[i], you must read all data blocks, and verify that all the data including data[i] is valid using the local hash.
        iii. To verify a given record, you must read all data blocks from the server and verify that the hash using data[i] matches the hash using the given record.

        All of the above methods require you to have more program memory than you have local storage.
    b. The cost of reads and writes in this system scales linearly with n.

6. There are two possible ways to use merkle trees for this. The first is to locally store all of the hashes of the records, the intermediate hashes, and the tree root. If this were possible, we could just store hashes of each record normally, so I assume this isn't the solution under discussion. The other option is to store remotely, in addition to the records, the merkle tree of all of the records, and to store the root of that tree locally. Perhaps the remotely stored merkle tree should be signed and given a timestamp that is also stored locally, so that it can be verified.
    a. To read data[i], you must request both data[i] and the merkle tree from the server. Verify that the merkle tree is correct using the locally stored root, and verify that data[i] fits within that tree.
    b. To write data[i], you request the merkle tree, verify that it is current and correct, compute its new values with the changed data[i], and then write both the new data[i] and the new merkle tree to the server. Store the new root locally.
    c. To verify a given record, you request the merkle tree, verify that it is current and correct, and check that the given record matches with the tree.
7.
    a. .15 * (number of blocks in a day) = .15 * (144) = 21.6 blocks/day on average
    b. Expected blocks found = (t/10 blocks) * (α chance per block) = α * t / 10
8. I answered both #8 and #9 from the perspective of the network.
   One block will be left orphaned whenever a node finds a block while an alternative block is already in transmission to it. The chance that, once a block is found, the other node finds a block within L seconds is $(1 – \alpha) * (L / 600)$. This chance occurs whenever a node finds a block, so multiplying it by the number of blocks per day gives $144 * \alpha * (1 – \alpha) * L / 600$. However, this applies for each of the nodes that can find a block. Because there are two nodes, α and $(1 – \alpha)$ are symmetric, so the final answer is:

   $$\text{Expected orphaned blocks per day} = 2 * \alpha * (1 – \alpha) * L * 144 / 600$$

9. I assume that both nodes start on an even footing. If the selfish miner has found zero or one blocks by the time the honest miner finds one, then there are no orphaned blocks, because the selfish miner doesn't release anything. If the selfish miner finds two blocks before the honest miner finds any, then the number of blocks they find before revealing will follow the distribution we discussed in class. The number of orphaned blocks will be one less than the number of blocks revealed in this case. The number of orphaned blocks could also equal the number revealed, if one or more additional blocks were found by the honest miner within the time L of the selfish miner's release. However that is a small chance (assuming L is small) and would require a disproportionately complicated analysis to take into account, so I'm going to ignore that chance here.

   The proportion of blocks which are orphaned under this system is the number of expected orphaned blocks from a release divided by the number of total blocks. The expected proportion of blocks mined honestly is $(1 – \alpha^2)$. The expected proportion of blocks mined by a selfish miner is $\alpha^2(2 + \alpha/(1 – 2\alpha))$. The proportion of blocks orphaned by a release is always one less than the number released selfishly, $\alpha^2(1 + \alpha/(1 – 2\alpha))$.

Proportion of blocks which will be orphaned: $$\frac{\alpha^2\left(1+\frac{\alpha}{1-2\alpha}\right)}{\left(1-\alpha^2\right)+\alpha^2\left(2+\frac{\alpha}{1-2\alpha}\right)}$$

Multiplying that by the expected number of blocks per day should give the expected number of orphaned blocks per day.

Expected number of orphaned blocks per day: $$144\times\frac{\alpha^2\left(1+\frac{\alpha}{1-2\alpha}\right)}{\left(1-\alpha^2\right)+\alpha^2\left(2+\frac{\alpha}{1-2\alpha}\right)}$$

Note: When the selfish node begins mining, there may be fewer than 144 blocks per day, because a large fraction of the hashing power has been removed from the honest chain. However, this will renormalize over time and return to the expected 144 blocks per day.

10. A:

A simple pool reward system that doesn't reward pool-hopping would be to give each miner a payout proportional to the number of shares they submitted since the last block the pool found, but with that value not mitigated by the total number of shares submitted. This way, the expected payout per time mining for each miner is constant. The pool can lose money if it takes a long time to find blocks on average, but that should only happen when mining power is leaving the pool and/or the bitcoin network. I think this is a real system called "Pay per Share".

Suppose the block reward were 100 BTC. A mining pool is composed of three miners. Miner A holds 5% of the total hashing power of the network, which is 17% of the pool's hashing power. Miner B holds 10% of the hashing power of the network (33% of the pool's) and Miner C holds 15% of the hashing power of the network (50% of the pool's). The pool operator expects it to take 10 minutes to find a block, and that each miner can submit 1 share per minute per 1% of the total network's hashing power that miner possesses. Therefore, the number of shares expected will be equal to the percentage of the total hashing power the pool controls. The pool controls 30% of the hashing power, so the payout for each share would be (100/30 * 30) = .1111… BTC (with no fee of any kind). It is important to divide by the total hashing power of the pool to account for the increased time between found blocks for the pool. This payout per share should be adjusted when the difficulty changes.

If it takes this pool 30 minutes to find a block since the last time it found one, then when it does, Miner A will have submitted 150 shares (for ~16.66 BTC), Miner B will have ~33.33 BTC, and Miner C will have ~50 BTC.

If the block had been found in 3 minutes instead, the payouts would be 1.66 BTC for A, 3.33 for B, and 5 for C.

The expected reward here only depends on the time since the last block was found insofar as more time gives miners the opportunity to do more work. The reward per share found doesn't change at all over time, because it was determined when the difficulty last changed and hasn't been altered since. Thus, there is no incentive to alter your mining habits based on the time since the last block was found.