A.J. Varshneya
CS 4501
9 October 2015

Problem Set 2

Problem 1

One assumption that we are operating under is that the miner is seeking to expand their wealth in the first place. Given this assumption, in order for it to be more profitable to mine than to defraud people, the expected reward for mining must exceed the expected value obtainable by defrauding people. Satoshi assumes that the coinbase transactions given to miners as a reward for verifying transactions has greater value than what a miner might be able to defraud of other bitcoin users.

Satoshi also assumes that if the community notices that this attacker is defrauding people, the wealth that he accrues will decrease in value. As the attacker has the most hashing power and is generating or stealing coin at the highest rate of any other node, he has the most to lose if the currency loses adoption and hence it is in their best interest to ensure everything is decentralized. We assume that if the attacker takes control of the bitcoin market, they will badly harm the bitcoin community and the coin they have accumulated becomes less valuable.

Problem 2

There is a 0.1% chance of the honest miners catching up to the attacker if he (or she) has 45% of the network's hashing power and a 340 block lead.

Problem 3

$p < 0.05$

q	z
0.10	3
0.15	3
0.20	5
0.25	6
0.30	10
0.35	17
0.40	36
0.45	137

Problem 4

a. If the bytes of data[i] don't indicate the record's position the server could have difficulty ensuring that the records are in order. Different records may be transmitted over different paths/networks from the user to the cloud storage so they may arrive out of order or possibly not at all. So we need some means of putting the record back in order when we send them to the cloud storage and also when we download the records when we read from the storage. Furthermore, we may not know if the server is actually returning the record from the index we query. They might return the jth record when we ask for the ith record. Including the index of each record as part of its data solves these issues.

b. One vulnerability is that we don't know necessarily if what we receive back is the newest version of our record. We might send multiple versions and since we don't store the hashes locally, we have no way of knowing if what we get back is the newest version. Otherwise, I think that since we used a private key to sign our messages, we can be fairly certain that what we read back is something that we at least sent. While an attacker might try to spoof a signature that works with the tampered data, they would be unable to without knowing our private key. When we read our data and signature back, we can use the data we receive back and our private key to recompute the signed message digest. If our data had been tampered with, the signed message digest would be very unlikely to match what we received back from the cloud storage.

Problem 5

a. To write, the user would start by generating a message digest for their data and storing it locally. They would then send the concatenated data to the cloud storage. To read/verify, the user would query the server to return their data, hash the record, and compare it to the digest they stored locally before they wrote. If these two hashes match, then the data is extremely unlikely to have been tampered with.

b. Writing a single record to the database involves hashing the entire concatenated message to store locally. We also must send the entire concatenated message over the network to cloud storage. So <u>writing</u> using this scheme scales <u>linearly</u> with the number of records, *n*. Reading/verifying requires receiving the entire concatenated message over the network, then again rehashing the entire concatenated message to compare it with our local hash. So <u>reading</u> also scales <u>linearly</u> with the number of records, *n*.

Problem 6

a. With this scheme, the server would maintain a merkle tree in cloud storage and the user would store the root of that tree locally. When the user first uses the cloud service, they would compute the root of this merkle tree and send their data. To write a new record, the user would first hash the record. They would then ask the server for the necessary hashes to compute the merkle root and verify that those hashes are accurate by computing the old merkle root with them. They would compute the new merkle root and send their data over the network where the server would store the data and update the merkle tree. To read/verify, the user would query the server to return their data along with the hashes necessary to compute the merkle root. The user would hash the data they received, use the given hashes to compute the merkle root, and verify the computed merkle root is what they had stored previously.

b. Writing to the database requires the user to hash the record and update their merkle tree root so it scales with *log(n)*. Reading/verifying a record requires the user to compute a merkle root to compare to the one they've stored locally, so it again scales with *log(n)*.

Problem 7

a. If a block is found every 10 minutes, there are 144 blocks found per day. Assuming the pool is not mining selfishly, they will discover 0.15*144 = 21.6 blocks per day.

b. (α blocks / cycle) * t minutes = (α blocks / 10 minutes) * t minutes
 = α*t/10 blocks

Problem 8

With a block found every 10 minutes, there are 144 periods in the day where we have latency between block discoveries. In 144α of those periods, the pool wins and in 144(1-α) of them another node in the network wins. The number of blocks mined in a period of latency by the pool is (α)(L/60)/10 = αL/600 and by the rest of the network it is (1-α)L/600. In total, we have 144α(1-α)L/600 + 144(1-α)αL/600 = (12/25)Lα(1-α) blocks mined in periods of latency.

Problem 9

If the mining pool is selfish, they will orphan $\alpha^2(1 + \alpha/(1-2\alpha))$ blocks per cycle from a zero-lead state. The total number of blocks per cycle from the initial state would be $\alpha^2(2 + \alpha/(1-2\alpha)) + (1-\alpha^2)$. So the proportion of orphaned blocks to all blocks is $[\alpha^2(1 + \alpha/(1-2\alpha))] / [\alpha^2(2 + \alpha/(1-2\alpha)) + (1-\alpha^2)]$. If a day has 144 cycles, we have in total 144 * $[\alpha^2(1 + \alpha/(1-2\alpha))] / [\alpha^2(2 + \alpha/(1-2\alpha)) + (1-\alpha^2)]$ orphaned blocks per day.

Problem 10

10b.

Professor

- What kind of taxes do I have to pay on bitcoin?
- If I run up someone's electric bill with an ASIC miner, can I be sued?

FBI Agent
- What sort of methods do you use to track down these criminals?
- If they’re in other countries, how do you prosecute them? Are you ever unable to?
- How can I protect myself from these ransomers?